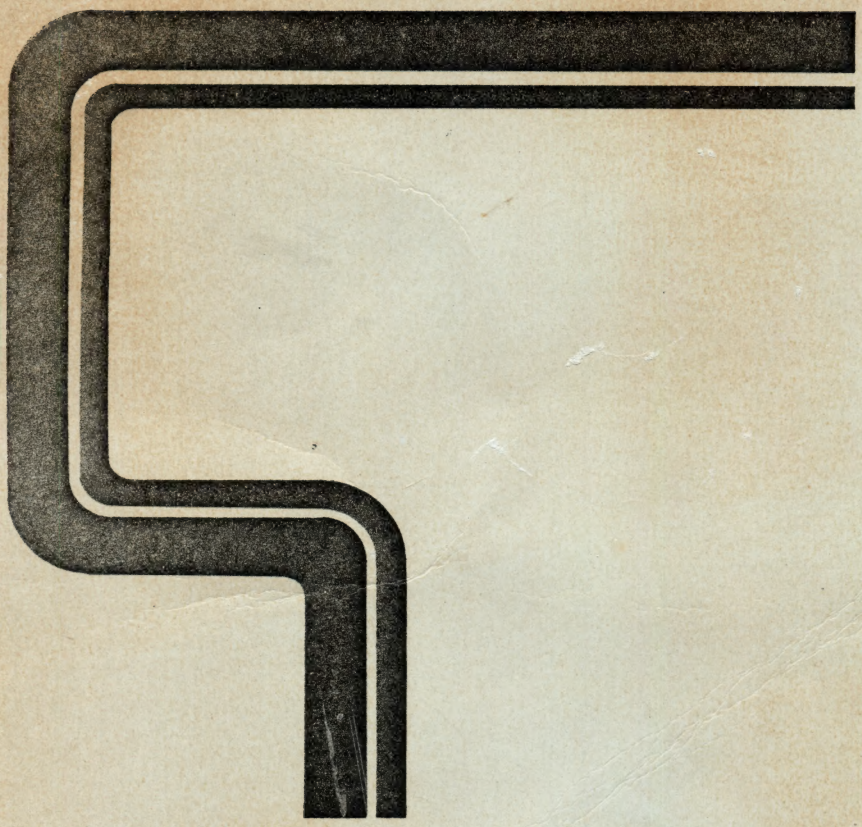
HARRY STEWART



MICROTEC

# MICRO COMPUTER SYSTEMS

# 6500
# RELOCATABLE MACRO ASSEMBLER MANUAL

Microtec
P.O. Box 60337
Sunnyvale, CA. 94088
408-733-2919

TABLE OF CONTENTS


| | | |
|---|---|---|
| 1.0 | INTRODUCTION | 1-1 |
| 2.0 | ASSEMBLER LANGUAGE | 2-1 |
| | Statements | 2-3 |
| | Comment Statement | 2-3 |
| | Special Symbols | 2-3 |
| | Special Characters | 2-4 |
| | Assembly Program Counter | 2-4 |
| | Symbolic Addressing | 2-5 |
| 3.0 | SYNTAX | 3-1 |
| | Character Set | 3-1 |
| | Symbols | 3-1 |
| | Constants | 3-2 |
| | Expressions | 3-4 |
| | Special Arithmetic Operators | 3-5 |
| | Indirect Addressing | 3-6 |
| 4.0 | DIRECTIVES | 4-1 |
| | Equate Dirctives | 4-1 |
| | ORG | 4-3 |
| | END | 4-4 |
| | EQU | 4-5 |
| | SET | 4-6 |
| | BYTE | 4-7 |
| | DATA | 4-7 |
| | WORD | 4-8 |
| | ACON | 4-8 |
| | DBYTE | 4-9 |
| | RES | 4-10 |
| | BSZ | 4-11 |
| | SKIP | 4-12 |
| | SPAC | 4-12 |
| | EJEC | 4-13 |
| | TITLE | 4-14 |
| | PAGE | 4-15 |
| | LIST | 4-16 |
| | NLIST | 4-18 |
| | IF | 4-19 |
| | ELSE | 4-20 |
| | ENDIF | 4-21 |

| | | |
|---|---|---|
| 5.0 | MACROS | 5-1 |
| | Macro Heading | 5-1 |
| | Macro Body | 5-1 |
| | Macro Terminator | 5-2 |
| | Macro Call | 5-2 |
| | LOCAL | 5-5 |
| | EXITM | 5-6 |
| | Macro Parameter Count | 5-7 |
| 6.0 | RELOCATION | 6-1 |
| | Program Segments | 6-1 |
| | Linking | 6-2 |
| | Relocatable Symbols | 6-2 |
| | Relocatable Expressions | 6-3 |
| | Zero Page Addressing | 6-4 |
| | Relocation Directives | 6-6 |
| | ASCT | 6-7 |
| | PSCT | 6-8 |
| | DSCT | 6-9 |
| | BSCT | 6-10 |
| | CSCT | 6-11 |
| | ORG | 6-12 |
| | XDEF | 6-13 |
| | PUBLIC | 6-13 |
| | XREF | 6-14 |
| | EXTRN | 6-14 |
| | XREFB | 6-15 |
| | NAME | 6-16 |
| | STKLN | 6-17 |
| 7.0 | HOW TO USE THE ASSEMBLER | 7-1 |
| | The Assembler | 7-1 |
| | Assembler Operation | 7-1 |
| | Assembler Listing | 7-2 |
| | The Object Module | 7-6 |
| | Cross Reference Format | 7-8 |
| | APPENDIX A - Assembler Error Codes | 8-1 |
| | APPENDIX B - ASCII and EBCDIC Codes | 8-4 |
| | APPENDIX C - 6500 Operation Codes | 8-5 |
| | APPENDIX D - Hexadecimal Notation | 8-10 |
| | APPENDIX E - Hexadecimal-Decimal Conversion Table | 8-11 |

# INTRODUCTION

MICROTEC has developed a Relocatable Macro Assembler for the 6500 series of microprocessors that translates Symbolic Machine Code into either absolute object code or relocatable object code which may then be processed by MICROTEC'S Linking Loader. The Assembler program is written in FORTRAN IV to achieve compatibility with most computer systems. It is modular and may be executed in an overlay mode when memory restrictions make that necessary. The program is approximately 4300 FORTRAN statements in length, 20% of which are comments. The program is written in ANSI standard FORTRAN IV and no facility peculiar to any one machine was utilized. This was done in order to eliminate FORTRAN compatibility problems.

The mnemonic Operation Codes as well as Assembler Directives are identical to those utilized by MOS Technology in their literature and in their software products. This has been done to eliminate compatibility problems and to prevent the necessity of learning new assembly languages.

The assembler is a two pass program that issues helpful error messages, produces an easily read program listing and symbol table, and outputs a computer readable absolute or relocatable object (load) module.

The assembler features relocation, macro capability, conditional assembly, symbolic and relative addressing, forward references, complex expression evaluation, cross reference listing and a versatile set of directives. Additionally, the assembler is capable of generating data in several number based systems as well as both ASCII and EBCDIC character codes. These features aid the programmer/engineer in producing well documented, working programs in a minimum of time.

MICROTEC does not present any information in this manual that will help the user understand the 6500 series of microprocessors, nor has any information been included to help the user write working programs. The reader is refered to the MOS Technology Cross Asssembler Reference Manual and Programming Manual.

# ASSEMBLER LANGUAGE

The assembler language provides a means to create a computer program. The features of the Assembler are designed to meet the following goals:

- Programs that are easy to create
- Programs should be easy to modify
- Programs should be easy to read and understand
- A machine readable load module to be generated

This assembler language has been developed with the following features:

- Symbolic machine operation codes (opcodes,directives)
- Symbolic address assignments and references
- Relative addressing
- Data creation statements
- Storage reservation statements
- Assembly listing control statements
- Address constants
- Character codes in ASCII or EBCDIC
- Comments and remarks for documentation
- Cross reference table listing
- Relocatable or absolute object module

An assembly language program is a program written in symbolic machine language. It is comprised of statements. A statement is either a symbolic instruction, a directive, a macro statement, or a comment.

The symbolic machine instruction is a written specification for a particular machine operation expressed by symbolic operation codes and sometimes symbolic addresses or operands. For example:

```
ISAM    LDA    MEM,X
```

where:

ISAM - is a symbol which will represent the memory address of the instruction.

LDA - is a symbolic op-code which represents the bit pattern of the "load instruction".

MEM - is a symbol representing a memory address.

X - is a reserved symbol representing the X index register and denoting it as an indexed instruction.

A directive statement is a statement which is not translated into a machine instruction, but rather is interpreted as a directive to the assembler program. For example:

```
ABAT    .WORD  DELT
```

where:

ABAT - is a symbol. The assembler is to assign the memory address of the first byte of the two allocated bytes to this symbol.

WORD - is a directive which directs the assembler program to allocate two bytes of memory.

DELT - is a symbol representing an address. The assembler is directed to place the equivalent memory address into the two allocated bytes.

## Statements

Statements are always written in a particular format as depicted below:

| LABEL FIELD | OPERATION FIELD | OPERAND FIELD | COMMENT FIELD |
|---|---|---|---|

The statement is always assumed to be written on an 80 column data processing card or as an 80 column card image.

The Label Field is provided to assign symbolic names to memory addresses or values. The Label Field may begin in any column if it is terminated by a colon or it may begin in column one with the colon omitted. A statement may consist of only a lable

The Operation Field is provided to specify a symbolic operation code, a directive, or a macro call. If present, this field must either begin beyond column one or be separated from the Label Field by one or more blanks or a colon. In the case of those instructions which may use accumulator mode addressing, i.e. ASL, LSR, ROL, ROR; the symbol A may be placed immediately after the opcode or be separated by one or more spaces or tabs. For example:

```
ASLA
LSR    A
```

The Operand Field is provided to specify arguments for the operation in the Operation Field. The Operand Field, if present, is separated from the Operation Field (including the A in LSR A) by one or more blanks or tabs.

The Comment Field is provided to enable the assembly language user to optionally place a message stating the purpose of a statement or group of statements. The Comment Field must be separated from the preceding field by one or more blanks or tabs or a semicolon.

A statement may be completely blank in which case a blank line will appear on the output listing.

## Comment Statement

A Comment Statement is a statement that is not processed by the assembler program. It is merely reproduced on the assembly listing. A comment statement is indicated by encoding an asterisk or a semicolon as the first nonblank character on a line. This may be in any column. If the first two characters on a statement are "*=", it will be interpreted as an Assembler Directive (see Section 4). An example comment statement is:

```
        ; THIS IS A COMMENT STATEMENT
```

If desired, logical columns 73-80 which are never processed by the assembler may be used for statement sequence numbers.

## Special Symbols

The assembler has internally defined the three symbolic register names that MOS Technology uses in their product descriptions. The symbols are:

A - accumulator
X - index register X
Y - index register Y

The user must use these special symbols in the operation or operand field of an instruction that requires them, but they may also be used like any other user defined symbol providing the symbol can be distinguished from its special usage. For example:

```
X       CLC
        -
        -
        BEQ    X
```

These special symbols will not appear on a symbol table or cross reference listing unless they are used in the label field of an instruction.

In addition, the following two symbols denote the "STACK" and "MEMORY" segments of a program (see Section 6).

STACK MEMORY

Finally the predefined symbol "NARG", described in Section 5, is used to represent the number of arguments passed in a macro call.

## Special Characters

Two special characters are defined by the assembler, "#", sharp sign, and "&", ampersand. The "#" is used to indicate the immediate mode of addressing for those opcodes that may use it. The "&" (also see special arithmetic operators in Section 3) may be used to force Zero Page addressing in either an absolute or relocatable assembly.

In general, when an instruction may be used with Zero Page or Absolute addressing, the assembler will use Zero Page if possible. However, if the programmer references a symbol that is as yet undefined in the program, the assembler does not know whether the instruction can be Zero Page, and hence, uses the Absolute mode of addressing. To avoid this problem, the programmer may use the "&" to indicate Zero Page addressing. For all symbols that previously defined, the assembler will automatically use Zero Page addressing, if possible, whichs reduces the user of the "&" symbol. Zero Page and Absolute addressing are discussed further in Section 6 on relocation. For example.

```
        LDA     DATA
        -
        -
        *=200
DATA    BYTE    4
```

The assembler will generates 3 bytes for the LDA instruction since it does not know that DATA may be accessed on the Zero Page. In addition the assembler will generate a warning flag which indicates that the instruction could have been Zero Page. condition. To alleviate this condition the programmer could write.

```
        LDA     &DATA
```

## Assembly Program Counter

During the assembly process the assembler maintains a variable that always contains the address of the next memory location to be assembled. This variable is called the Assembly Program Counter. It is used by the assembler to assign addresses to assembled bytes, but is also available to the programmer.

The character "*" is the symbolic name of the Program Counter. It may be used like any other symbol, but it may not appear in the label field (except for certain directives).

When using the "*", the programmer may think of it as expressing the idea; "*" = "address of myself." For example:

```
10      JMP     *
```

The jump instruction is in location 10. The instruction directs the microprocessor to "jump to myself." The Program Counter in this example contains the value 10 and the instruction will be translated to a jump to location 10 from location 10. This could be used when waiting for an interrupt.

## Symbolic Addressing

When writing statements in symbolic machine language, i.e assembler language, the machine operation code is usually expressed symbolically. For example the machine instruction that rotates the contents of the accumulator may be expressed as:

```
ROL     A
```

When translating this symbolic operation code and its arguments into machine language for the 6500, the assembler defines one byte containing 2AH at the memory location indicated by the current Assembly Program Counter. The address of the translated byte is known because the Assembly Program Counter is always set to the address of the byte currently being assembled.

The user can optionally attach a label to such an instruction. For example:

```
SAVR    ROL     A
```

The assembler, upon seeing a valid symbol in the label field, assigns the equivalent address to the label. The equivalent addresses is the address contained in the Assembly Program Counter. In the given example, if the ROL instruction is to be stored at the address 127, then the symbol SAVR will be set to the value 127 for the duration of the assembly.

The symbol could then be used anywhere in the source program to refer to the instruction location. The important concept is that the address of the instruction need not be known; only the symbol need be used to refer to the instruction location.

Thus when jumping to the ROL instruction, the user could write:

```
JMP     SAVR
```

When the jump instruction is translated by the assembler, the address of the ROL instruction is placed in the address field of the instruction.

It is also possible to use symbolic addresses which are near other locations to refer to those locations without defining new labels. This may be done through use of the + and - operators. For example:

```
        BPL     BEG+4
        JSR     *+48
BEG     LDA     DATA
        CPX
        ASL     A
        INC     PAUL
```

In the above example, the instruction "BPL BEG+4" refers to the "INC" instruction. BEG+4 means the address of BEG plus 4 bytes.

# SYNTAX

The Assembler Language is a language like any other. That is, it has a character set, vocabulary, rules of grammar, and allows for individuals to define new words or elements. The rules that describe the language are termed the syntax of the language.

For a symbol, expression, or statement in assembler language to be translated by the assembly program, it must be written correctly in accord with the rules of syntax described below.

## Character Set

The following list of characters are those the assembler will recognize. They are the only valid characters and any others except in a comment field will generate an error. Many of the special characters have no previously defined meaning except as a character constant.

### Alphabetic Characters

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

### Numeric Characters

0 1 2 3 4 5 6 7 8 9

### Special Characters

| | | | |
|---|---|---|---|
| | blank | / | slash |
| > | greater than | $ | dollar |
| < | less than | * | asterisk |
| ' | single quote | ( | left parenthesis |
| , | comma | ) | right parenthesis |
| + | plus sign | @ | commercial at |
| - | minus sign | . | period |
| & | ampersand | : | colon |
| ! | exclamation | ; | semi-colon |
| " | double quote | = | equal sign |
| # | sharp | ? | question mark |
| % | percent | _ | underbar |

## Symbols

A symbol is a sequence of characters. The first character in a symbol must be alphabetic or one of the special characters: dollar sign, commercial at sign, or underbar. Except for the above three special characters none may

be used in a symbol. Imbedded blanks are not permitted. Even though the assembler generates "local symbols" that start with a question mark theyh should be avoided by the user.

Only the first six characters of a symbol are used by the Assembler to define that symbol; the remaining characters are for documentation. The parameter that dictates the number of characters used to define a symbol may be changed in the Assembler.

The Assembler´s symbol table may contain 200 symbols. This number may also be increased by changing a parameter in the Assembler.

Symbols are used to represent arithmetic values, memory addresses, bit arrays (masks), etc. Examples of valid symbols:

```
LAB1
MASK
LOOP$NUM              (symbol used is LOOP$N)
```

Examples of invalid symbols:

```
ABORT*                (contains special character)
1LAR                  (begins with a numeric)
PAN N                 (embedded blank, symbol is PAN)
```

Constant

A constant is an invariant quantity. It may be an arithmetic value or a character code. There are several ways of specifying constants in this assembler language.

Decimal constants may be defined as a sequence of numeric characters optionally preceded by a plus or a minus sign. If unsigned, the value is assumed to be positive.

In most cases, constants must be contained in one or two bytes. A one byte constant can contain an unsigned number with a value from 0 to 255. A two byte unsigned number can range from 0 to 65535. When a constant is negative, its equivalent two´s complement representation is generated and placed in the field specified. A one byte two´s complement negative may range from -1 to -256. A two byte two´s complement negative number may range from -1 to -65536.

All constants are evaluated as 16 bit quantities, i.e. modulo 65536. Whenever an attempt is made to place a constant in a field for which it is too large, an error message is generated by the assembler.

Other constants are defined by utilizing a coded descriptor after the constant or special characters before the constant like by MOS Technology.

The following list indicates the available descriptors and their meaning. If no descriptor is given, the number is assumed to be decimal.

B - binary
O - octal
Q - octal
D - decimal
H - hexadecimal

Using the above descriptors, a leading zero must be added to hexadecimal constants that begin with A-F.

Constants may also be defined by utilizing the following special characters which may only precede the constant. Hexadecimal constants that start with A-F are not required to have a leading 0 when utilizing these descriptors.

% - binary
@ - octal
$ - hexadecimal

Since octal constants may start with a "@", the user is cautioned about using symbols that start with a "@". Although the symbol will be valid in the label field, it will cause an error when used as an operand because the symbol will be processed as an octal constant.

Examples of these constants are:

| | | | | |
|---|---|---|---|---|
| 10011B | 25 | 0FFH | 1377Q | 255D |
| %1001 | @56 | 1777770 | $3AB | 45 |

The ASCII character constant is another form that may be specified. An ASCII character may be defined by a single quote followed by the character. For example.

```
ADC    #'A
```

An ASCII or EBCDIC character constant may be specified by enclosing a single character within quote marks and preceding it with an A for ASCII or an E for EBCDIC. If no descriptor is specified, the string is assumed to be ASCII. Examples of this constant form are;

```
ADC    #'Z'
EOR    #E'1'
AND    #A'0'
```

A character string may be specified by using the BYTE or DATA directive. Character strings must follows the format described for these directives (see Section 4). Characters in a string may be specified as ASCII or EBCDIC in a manner similar to the character constant. In these directives a single character or string must be enclosed within quotes. The directive, BYTE 'Q, will generate an error. Examples of the character string are:

```
A'TELETYPE CODES'
E'TERMINAL CODES'
' 123.7'
```

Note that one byte of memory is allocated for each character in a string. When a string is specified in a DATA or BYTE directive, characters are stored in sequential bytes of memory beginning at the first avalable byte.

To generate the code for a single quotation mark in the character constant or string it must be specified as two single quote marks. Example:

```
'DON''T'
```

The code for a single quote mark will be generated once for every two quote marks that appear contiguously within the character string.

## Expressions

An expression is a sequence of one or more symbols, constants or other expressions separated by arithmetic operators. Expressions are evaluated left to right subject to the precedence of operators shown below. Parentheses may be used to establish the correct order of the arithmetic operators and it is recommended they be used in complex expressions involving operators such as SHR, AND, EQ, etc. The operators and their precedence are:

| Precedence | Operator | |
|---|---|---|
| 1 | + | (unary plus) |
| | - | (unary minus) |
| 2 | * | (multiplication) |
| | / | (division) |
| | .MOD. | (modulo) |
| | .SHR. | (shift right) |
| | .SHL. | (shift left) |
| 3 | + | (addition) |
| | - | (subtraction) |
| 4 | .NOT. | (logical NOT) |
| 5 | .AND. | (logical AND) |
| 6 | .OR. | (logical OR) |
| | .XOR. | (exclusive OR) |

| | | |
|---|---|---|
| 7 | .EQ.,.NE. | (equality,not equality) |
| | .GT.,.GE. | (greater than) |
| | .LT.,.LE. | (less than) |
| 8 | .LOW. | (low 8 bits) |
| | .HIGH. | (high 8 bits) |

The comparision operators (.EQ., .GT., etc) return a logical True (all one bits) if the comparision is true and a logical False (zero) if the comparision is not true. All operands are considered to be unsigned 16 bit values.

The shift operators (.SHR., .SHL.) shift the argument that goes before the operator right or left the number of bits specified by the argument that comes after the operator. (Zeros are shifted into the high or low order bits.) shifted into the high or low order bits.

The .HIGH. and .LOW. operators have been provided to help the user define two byte addresses as individual bytes whenever desired. result of applying either of these operators is a one byte value which is unary and which may be used anywhere in an expression. When .HIGH. or .LOW. are used in a relocatable expression, the result will remain relocatable. This enables the user to relocate 8 bit values.

An expression must resolve to a single unique value. Consequently, character strings are not permitted in expressions. All expressions are evaluated modulo 65536. Whenever an attempt is made to place an expression value in a one byte field and the expression is too large, an error message is generated. Examples of valid expressions:

```
PAM+3                          LOOP+(ADDR.SHR.8)
(PAM+45H)/CAL                  VAL1.EQ.VAL2
IDAM.AND.255                   .LOW.EXPR
```

Embedded blanks may not exist within an expression

## Special Arithmetic Operators

The special characters, ">", greater than and, "<", less than, have been assigned as special arithmetic operators. They correspond to the operators .LOW. and .HIGH. and are used to obtain the lower and upper 8 bits of an expression respectively.

These operators may also be used to force an instruction to be Zero Page addressing but only if the operand contains no undefined symbols during pass 1. This is the case because >(expression) will always resolve to a value that may be contained in 8 bits.

## Indirect Addressing

For those instructions that may use the Indirect Addressing modes, the assembler assumes that Indirect Indexed or Indexed Indirect addressing is specified when the first character in the argument field is a left parenthesis. Since expressions may also contain parentheses, a potential conflict exists. For example:

```
LDA     (MEM+2)*4
```

This is not an Indirect instruction, but will seem like one to the assembler (it will finally generate an error because of the format). To avoid this problem, the user should precede any expressions of this type with a zero term as follows.

```
LDA     0+(MEM+2)*4
```

or

```
LDA     +(MEM+2)*4
```

# DIRECTIVES

The assembler directives are written as ordinary statements in the assembler language, but rather than being translated into equivalent machine language, they are interpreted as commands to the assembler itself.

Through use of these directives, the Assembler will reserve memory space, define bytes of data, assign values to symbols, control the output listing, etc.

This section describes all directives except those primarily associated with macro assembly and relocation. Some directives such as ORG apply to both absolute and relocatable assembly and are discussed in both sections.

To retain compatibility with MOS Technology's formats, all directives may be optionally preceded with a period. For example:

```
.BYTE
```

In addition, those directives defined by MOS Technology may be abbreviated to three characters. I.e. .BYT, .WOR, .DBY, .PAG, .SKI.

## Equate Directives

To retain compatibility with MOS Technology, equate directives may also be denoted by the equal sign. These directives are equivalent to the EQU, ORG, and RES directives as follows.

```
=                (EQU)
*=               (ORG)
*=*+expression   (RES)
```

When character "*" is followed immediately by an equal sign, it will not be treated as a comment line but rather as one of the above directives.

The directives described in this section are:

| | |
|---|---|
| ORG | Set Program Origin |
| END | End of Assembly |
| EQU | Equate a Symbol to an Expression |
| SET | Equate a Symbol to an Expression |
| BYTE | Data Definition |
| DATA | Data Definition |
| WORD | Define Address Constant |
| ACON | Define Address Constant |
| DBYTE | Define Reverse Double Byte |

| | |
|---|---|
| RES | Reserve Storage |
| BSZ | Block Storage with zero fill |
| SKIP | Space lines on listing |
| SPAC | Space lines on listing |
| EJEC | Advance Listing Form to Next Page |
| TITLE | Set Program Heading |
| PAGE | Advance Form and set Heading |
| LIST | List the Elements Specified |
| NLIST | Don´t List the Elements Specified |
| IF | Conditional Assembly Statement |
| ELSE | Conditional Assembly Converse |
| ENDIF | End Conditional Assembly |

In the following descriptions, brackets, { }, are used to indicate optionality. If more than one item appears within the same pair of brackets, a choice is indicated.

ORG - Set Program Origin (non-relocatable mode)
*=

The ORG directive is used to inform the assembler of the memory address to which the next assembled byte should be assigned. All subsequent bytes will be assigned sequential addresses beginning with this address.

If the program does not have an ORG as the first statement, an "ORG 0" is assumed and assembly begins at location zero in the absolute program segment. That is, the program will not be relocatable.
Example:

```
ORG    100H
```

| | | |
|---|---|---|
| {label} | ORG | expression |
| {label} | *= | expression |

where:

label - is an optional label which if present will be equated to the given expression.

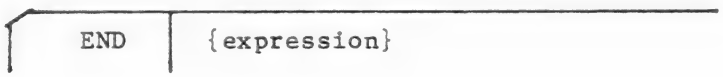
expression - a value which will replace the contents of the Assembly Program Counter and bytes subsequently assembled will be assigned memory addresses beginning with this value. Any symbols used in this expression must be previously defined. When using *=, the "*" and "=" must be adjacent characters. Blanks may follows the equal sign.

END - End of Assembly

The END directive is used to inform the assembler that the last source statement has been read and to indicate a load module starting address. Any statements following the END directive will not be processed.

Example:

```
        END     MAIN
```




where:

expression - is an address that is placed in the end record of the load module and informs the loader where program execution is to begin. Specifying a load address in this directive also informs the loader that this is a main program. If multiple load modules are combined by the Linking Loader, only one module may specify a load address and hence be a main program.

EQU - Equate a Symbol to an expression
=

The EQU or = directive causes the assembler to assign a particular value to a new label. This value may be an absolute value or a relocatable segment value (see Section 6).

Example:

```
SEVEN   EQU    7
MASK    =      10011B
```

| | | |
|---|---|---|
| label | EQU | expression |
| label | = | expression |

where:

label - is a symbol defined by this statement.

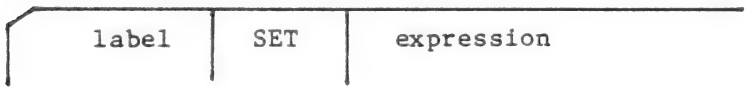
expression - is an expression whose value will be assigned to the given label for the duration of the current assembly. An attempt to reequate the same label will result in an error. Any symbols used in the expression must be previously defined. An external symbol may not be used in the expression. Blanks may exist on both sides of the equal sign.

SET - Equate a Symbol to an Expression

The SET directive sets a symbol equal to a particular value. But unlike the EQU directive, multiple SET directives for the same symbol may be placed in a source program. The most recent SET directive determines the value of the symbol until another SET is he source processed.

Example:

```
GO      SET     5
GO      SET     GO+10
```




where:

label - is a symbol defined by this statement

expression - is a value that will be assigned to the given label until changed by another SET directive. Any symbols used in the expression must be previously defined. An external symbol may not be used in the expression.

BYTE - Data Definition
DATA

The DATA or BYTE directive is used to define up to 70 bytes of data. The assembler will allocate one byte if an expression is given and will allocate several bytes if a character string is specified. All expressions must evaluate to an one byte value or an error is generated. Negative values are stored using their two's complement representation. A relocatable expression must be used with the .HIGH. or .LOW. operators. If neither operator is present an error is generated and the .LOW. operator is assumed. A relocatable expression which is in the Base Segment (Zero Page) or which is declared as a Base Segment External is valid without the use of the above operators.

Null operands separated by commas will have zeros stored as the operands.

Example: (with generated bytes shown)

```
7A 11 00        ITEM    DATA    +122,17,
06 1F 31                BYTE    6,1FH,'A'+1,32Q
1A
```

| {label} | BYTE<br>DATA | operand1,{operand2}, ... |
|---|---|---|

where:

label - is an optional label which will be assigned the address of the first byte defined.

operand - an evaluatable expression contained in one byte, a character constant or an ASCII or EBCDIC character string of up to 70 characters. In addition, the operand may be null, e.g. BYTE ,,; in which case a zero will be stored. All character strings must be enclosed within single quotes. A character preceded by a single quote only will generate an error, e.g. 'A.

WORD - Define Address Constant
ACON

These directives inform the assembler to allocate two bytes per operand. Each operand is stored in successive pairs of bytes. The operands are stored with the low order 8 bits in the first byte and the high order 8 bits in the second byte. Negative values are stored using their two's complement representation.

Null operands separated by commas will have zeros stored as their value.

Example: (will generated bytes shown)

```
00 1B 00        ADD1    ACON    1BH,$24+4
28 00 00
03 E8 27                WORD    1000,10000
10
```

| {label} | WORD<br>ACON | operand1,{operand2}, ... |
|---|---|---|

where:

label - is an optional label which will be assigned a value equal to the address of the first byte allocated by this statement.

operand - is an evaluatable expression contained in two bytes. A total of 70 bytes may be allocated by this directive. A null operand will cause two bytes of zero to be stored.

DBYTE - Define Double Byte

The DBYTE directive is similar to the WORD directive except for the order in which the 16 bit value of each operand is stored. The low order 8 bits of each operand is stored in the second byte (high address) of the double byte and the high order 8 bits are stored in the first byte. Negative values are stored using their two's complement representation.

Null operands separated by commas will have zeros stored as their value.

Important!! this directive should not be used with relocatable operands as the final result will cause errors at load time.

Example: (with generated bytes shown)

```
E8 03 10        REV1    .DBY   1000,10000
27
```

| {label} | DBYTE | operand1,{operand2}, ... |
|---|---|---|

where:

label - is an optional label which will be assigned a value equal to the address of the first byte allocated by this statement.

operand - is an evaluatable expression contained in two bytes. A total of 70 bytes may be allocated by this directive. A null operand will cause two bytes of zero to be stored.

RES - Reserve Storage
*=*

These directives are used to reserve a block of sequential bytes of storage. This directive merely causes the program counter to be advanced and hence, the contents of the reserved bytes are unpredicatable.

Example:

```
PAT     RES    62H
        *=*+@20
```

| {label} | RES<br>*= | expression<br>*+expression |
|---|---|---|

where:

label - is an optional label which will be assigned the address of the first byte allocated.

expression - a value which specifies the number of bytes to be allocated by this directive. Any symbols used in this expression must be previously defined. The final expression may not contain any relocatable terms. When using the *=* format, the "*=" must be adjacent. Blanks or tabs may exist between the "*=" and "*+expression".

BSZ - Block Storage will zero fill

The BSZ directive causes the assembler to allocate a block of storage similar to the RES directive with each byte being initialized to zero.

Example:

```
TABLE   BSZ     100
```

| {label} | BSZ | expression |
|---|---|---|

where:

label - is an optional label which will be assigned the address of the first byte allocated.

expression - a value which specifies the number of bytes to be allocated by this directive and whose value will be set to zero. Any symbols used in this expression must be previously defined and the expression may not be relocatable.

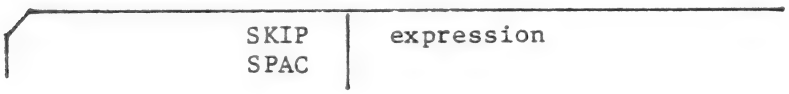
SKIP - Space lines on listing
SPAC

These directives cause one or more blank lines to appear on the output listing. It enables the programmer to format the program listing for easier reading. The directive itself does not appear on the program listing.

The user may also use a blank source statement to insert blank lines on the program listing.

Example:

```
SKIP   7
```




where:

expression - evalutes to a value which determines how many lines are to be skipped. This value may not be relocatable.

EJEC - Advance Listing Form to next Page

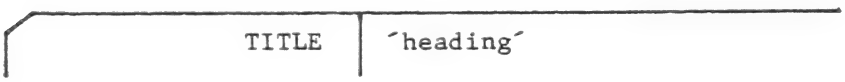
This directive instructs the assembler to skip to the top of the next page on the listing form. Its purpose is to make program listings easier to read. Some programmers prefer to start each subroutine on a new page.

TITLE - Set Program Heading

This directive is used to print a heading at the beginning of each page of the listing. The default heading defined by the assembler is: "6500 ASSEMBLER VER _._MR." For a user specified title to appear on the first page of the output listing, this directive must be the first statement in the program.

Example:

```
TITLE  'TEST PROGRAM'
```

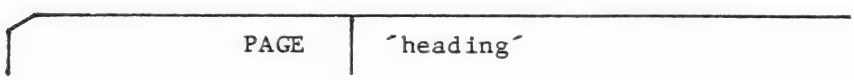



where:

heading - title which will be placed at the beginning of each page. Up to 50 characters may be used in the heading with additional characters being ignored. The heading is delimited by single quotes, but if the terminating quote is not present, the first 50 characters will be used. The heading may contain no characters in which case the title will be set to blanks.

PAGE - Set Program Heading and Advvance Listing Form

The PAGE directive causes the assembler to advance the listing form to the top of the next page similar to the EJEC directive. In addition, it may optionally set a title similar to the TITLE directive.




where:

heading - title which will be placed at the beginning of each page. The heading is similar to that described for the TITLE directive. If a heading is not specified, this directive is similar to the EJEC directive.

LIST - List the Elements Specified

The LIST directive may be used to generate listings of the elements specified. The defaults in the assembler are: the source text, symbol table, macro expansions, and conditional assembly statements not assembled are all listed. An object module in relocatable format is produced and the symbol table is not placed into the object module.

Errors messages are always listed regardless of the elements specified. In particular, the E option may be used to generate in a separate file, a listing consisting only of error messages.

Example:

```
LIST   X,B      list cross reference
                table and put symbol
                table in object module
```

| LIST | {A,B,E,G,I,M,O,S,T,X} |
|---|---|

where:

- A - specifies that this will be an absolute assembly. All relocation directives are flagged as errors and the object module is MOS Technology's hexadecimal format. This option should only be turned on, and only at the start of the program.
- B - specifies that the symbol table will be placed into the object module, either absolute or relocatable, and may be used for debugging.
- E - specified that errors will also be listed to a separate error file. This is useful if the user wants the normal output listing to say go to a line printer and the errors only to appear at a terminal. If only errors are desired, the user may turn the source listing off and ony the errors will appear on the normal output device.
- G - specifies any system generated symbols (see Section 5) will be listed in the symbol or cross reference table and in the object module (if B specified).
- I - specifies that the instructions not assembled due to conditional assembly statements be listed (default).
- M - specifies that expanded macros will be listed in the program listing (default).
- O - specifies that the object module will be produced (default).
- S - specifies the source text will be listed (default).

T - specifies the symbol table will be listed (default).
X - specifies that the cross reference table will be listed. This parameter overrides the T option if specified. Thus if both T and X are specified, a cross reference table will be generated. (see page 7-8).

NLIST - Suppress Listing of the Elements Specified

The NLIST directive instructs the assembler to suppress the listings of the elements specified. The listings may be enabled again by the LIST directive. Errors generated by the assembler are always listed regardless of the list flags. Thus to obtain an output which lists only errors, the user should specify "NLIST S" at the beginning of the program. The user may also use the "E" option to obtain errors only.

Example:

```
        NLIST     O         do not produce an
                            object module
```

| | |
|---|---|
| NLIST | {A,B,E,G,I,M,O,S,T,X} |

where:

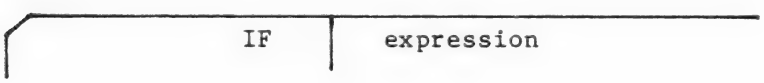
- A - specifies that the assembly is relocatable. This option should not be used as the option should only be turned on.
- B - specifies that the symbol table will not be placed into the object module.
- E - specifies that errors will not be listed in the error file.
- G - specifies that system generated symbols will not be listed in the symbol table or object module.
- I - specifies that the instructions not assembled due to conditional assembly statements not be listed.
- M - specifies that expanded macros not be listed.
- O - specifies that the object module will not be produced.
- S - specifies that the source text will not be listed. Only those statements will errors will be listed.
- T - specifies that the symbol table will not be listed.
- X - specifies that a cross reference table will not be produced or listed.

IF - Conditional Assembly Statement

The IF directive may be used to conditionally assemble source text between the IF directive and the ELSE or ENDIF directive. When the expression in the operand field evaluateds to an non-zero value, the code will be assembled. If the expression evaluates to a value of zero, the code will not be assembled. IF statements may be nested up to 16 levels and appear at any place within the source text.

Example:

```
                    IF      SYSTEM
```




where:

expression - evaluates to a value which determines whether or not the assembly between the IF and the following ELSE or ENDIF will take place. Any symbols used in this expression must be previously defined. The expression may not be relocatable.

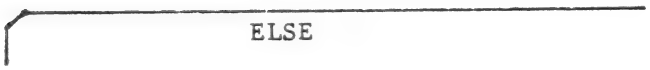
ELSE - Conditional Assembly Converse

The ELSE directive is used in conjuction with the IF directive and is the converse of the IF. When the expression of the IF directive evaluates to zero, all statements between the ELSE directive and the next ENDIF are assembled. When the argument of the IF directive evaluates to a non-zero value all statements between the ELSE directive and the next ENDIF are not assembled.

The ELSE directive is optional and can only appear once within an IF-ENDIF block.

Example:

```
IF      MAIN
-
ELSE
-
ENDIF
```

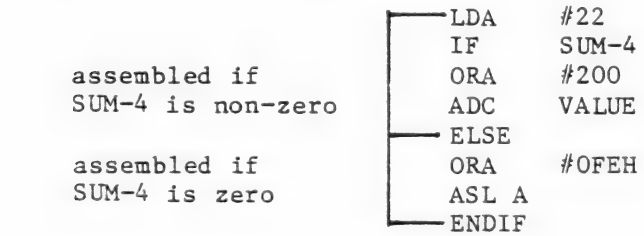


ENDIF - End Conditional Assembly Code

The ENDIF directive is used to inform the Assembler where the source code subject to the conditional assembly statement ends. In the case of nested IF statements, an ENDIF is paired with the most recent IF statement.

Example:

In the following code, if the expression SUM-4is equal to zero, the instructions between the IF and ELSE directives will not be assembled and those between the ELSE and ENDIF will be assembled. If SUM-4 is non-zero, the opposite occurs. To inhibit listing the non assembled instructions the "NLIST I" directive may be used.



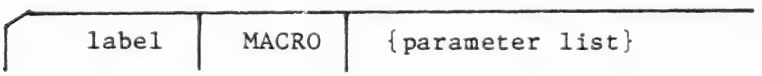
# MACROS

A macro is a sequence of instructions that can be automatically inserted in the assembly source text by encoding a single instruction, the macro call. The macro definition is written only once and can be called any number of times. The macro definition may contain parameters which can be changed for each call. The macro facility simplifies the coding of programs, reduces the chance of programmer error, and makes programs easier to understand as the source code need only be changed in one location, the macro definition.

A macro definition consists of three parts; a heading, a body, and a terminator. This definition must precede any macro call to this macro. A macro may be redefined at any place in the program. The most recent definition of a macro name will be used when the macro is called. A standard assembler mnemonic (e.g. ORA) may also be redefined by defining a macro with the name ORA. In this case all subsequent uses of the ORA instruction in the program will cause the macro to be expanded.

## Macro Heading

The heading, which consists of the directive MACRO, gives the macro a name and defines any formal parameters.




Label specifies the macro name and may be any user defined symbol. This name may be the same as other program defined symbols since it has meaning only in the operation field. For example, TAB could be the name of a symbol as well as a macro.

If a macro name is identical to a machine instruction or an assembler directive, the mnemonic is redefined by the macro. Once a mnemonic has been redefined as a macro, there is no way of returning that name to be a standard instruction mnemonic. A macro name may also be redefined as a new macro with a new body.

The operand field of the MACRO line contains the name of dummy formal parameters in the order in which they occur on the macro call. Each parameter is a symbol and multiple parameters must be separated by commas. The symbols used as formal paramters are known only to the macro definition and may be used as regular symbols outside the macro.

## Macro Body

The first line of code following the MACRO directive which is not a LOCAL

directive is the start of the macro body. These statements are placed in a macro file for use when the macro is called. During a macro call, an error will be generated if another macro is defined within a macro. No statements in a macro definition are assembled at definition time, they are simply stored in the macro file until called, at which time they are inserted in the source code at the position of the macro call.

The name of a formal parameter specified on the MACRO directive may appear within the macro body in any field. If a parameter exists, it is marked, and the real corresponding parameter from the macro call will be substituted when the macro is called. Parameters are not recognized in a comment statement or the comment field of a statement.

## Macro Terminator

The ENDM directive terminates the macro definition. During a Macro definition an ENDM must be found before another MACRO directive may be used. An END directive that is found during a macro definition will terminate the macro definition as well as the assembly. The format of the ENDM directive is as follows:

| {label} | ENDM |
|---|---|

where:

label - is an optional label which becomes the symbolic address of the first byte of memory following the inserted macro.

## Macro Call

A macro may be called by encoding the macro name in the operation field of the statement. The format of the call is shown below.

| {label} | name | {parameter list} |
|---|---|---|

where:

label - is an optional label which will be assigned a value equal to the current program counter.

| | |
|---|---|
| name | - is the name of the macro called. This name should be defined by the MACRO directive or an error message will be generated. |
| parameter list | - is a list of parameters separated by commas. These parameters may be constants, symbols, expressions, character strings or any other text separated by commas. |

The parameters in the macro call are actual parameters and their names may be different than the formal parameters used in the macro definition. The actual parameters will be substituted for the formal parameters in the order in which they are written. Commas may be used to reserve a parameter position. In this case the parameter will be null, that is contain no actual characters. The formal parameter corresponding to a null actual parameter is simply removed during macro expansion. Any parameters not specified will also be null. The parameter list is terminated by a blank, tab, or semicolon.

All actual parameters are passed as character strings into the macro definiton statements. Thus symbols are passed by name and not by value. In other words, the parameters are not evaluated until the macro expansion is produced. Thus SET directives within a macro body may alter the value of parameters passed to the macro.

During the macro expansion, the assembler recognizes certain characters to have special meaning. The ampersand "&", is used to concatentate the text of the definition line and any actual parameters. During macro expansion, an ampersand immediately preceding or immediately following a formal parameter is removed and the substitution of the actual parameter occurs at that point. If the ampersand is not immediately adjacent to the parameter, the ampersand is not removed and remains part of the definition line. Ampersands within character strings are not recognized as concatenation operators.

The angle brackets, "< >", are used to delimit actual parameters that may contain other delimiters. When the left bracket is the first character of any parameter, all characters between it and the matching right bracket are considered part of that parameter. The outer brackets are removed when the parameter is substituted in a line. Angle brackets may be nested for use within nested macro calls. The brackets are the only way to pass a parameter that contains a blank, comma, or other delimiter. For example, to use the instruction "ASL A" as an actual parameter would require placing <ASL A> in the actual parameter list. A null parameter may consist of the angle brackets with no intervening characters.

An example of a macro call and its expansion is shown below. Note that expanded code is marked with plus signs.

Definition:

```
        GET     MACRO   W,Y,Z
                LDA     #W
                ASL     A
                Y
        Z       BEQ     MAIN
                ADC     #-5
                ENDM
```

Call:

```
                -
                CLC
        LOOP    GET     200,<STA    DATA>,ENTRY
                JMP     FIRST
                -
```

Expanded Macro:

```
                -
                CLC
        LOOP    GET     200,<STA    DATA>,ENTRY
      +         LDA     #200
      +         ASL     A
      +         STA     DATA
      +ENTRY    BEQ     MAIN
      +         ADC     #-5
                JMP     FIRST
```

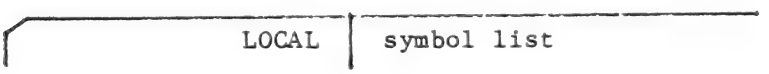
LOCAL - Define Local Symbol

Because all labels, including those within macros, are global, that is known, to the entire program, a macro which contains a label and which is called more than once will cause a duplicate label error to be generated. To avoid this problem, the user may declare labels within macros to be "local" to the macro. Each time the macro is called the assembler assigns each local symbol a system generated unique symbol of the form ??nnnn. Thus the first local symbol will be ??0000, the second ??0001, etc. The assembler does not start at ??0000 for each macro but increases the count for each local symbol encountered. The symbols defined in this directive are treated like formal macro parameters and hence may be used in the operand field of instructions. The operand field of the LOCAL directive may not contain any formal parameters defined on the MACRO directive line. As many LOCAL directives as necessary may be included within a macro definition but they must occur immediately after the MACRO directive and before the first line of the macro body. LOCAL directives that appear outside a macro definition will generate an error. For example:

```
Definition:     WAIT    MACRO   TIME
                        LOCAL   LAB1
                        LDY     #TIME
                LAB1    DEY
                        BNE     LAB1
                        ENDM


First call      +       LDY     #5
with TIME=5     +??0000 DEY
                +       BNE     ??0000

Second call     +       LDY     #$FF
with TIME=$FF   +??0001 DEY
                +       BNE     ??0001
```




where:

symbol list - is a list of symbols separated by commas that are to be defined local to this macro.

EXITM - Alternate Macro Exit

The EXITM directive provides an alternate method for terminating a macro expansion. During a macro expansion, an EXITM directive causes expansion of the current macro to stop and all code between the EXITM and the ENDM for this macro to be ignored. If macros are nested, EXITM causes code generation to return to the previous level of macro expansion. Note, an EXITM or and ENDM may be used to terminate a macro expansion, but only an ENDM may be used to terminate a macro definition.

In the following example, the code following the EXITM will not be assembled if DATA is non zero.

```
STORE   MACRO  DATA
         -
         -
        IF     DATA
        EXITM
        ENDIF
         -
         -
        ENDM
```

| {label} | EXITM |
|---|---|

where:

label - is an optional label which will be given the address of the current program counter.

## Macro Parameter Count

The special symbol "NARG" may be used by those users who need to know the number of paraemters passed on the macro call statement to the macro. This symbol is used like any other symbol and represents the number of parameters passed to the macro, not defined in the macro definition. The symbol should only be used within a macro although its value is set to zero outside of a macro. This symbol is typically used when generating tables within macros along with conditional assembly. This count only represents parameters which are not null. For example:

```
Definition:     GEN     MACRO   P1,P2,P3
                        IF      NARG
                        BYTE    P1
                        GEN     P2,P3
                        ELSE
                        ENDIF
                        ENDM

Macro Call:             GEN     ADD1,ADD2

Expansion:      +       IF      2 (value of NARG)
                +       BYTE    ADD1
                +       GEN     ADD2,
                +       IF      1
                +       BYTE    ADD2
                +       GEN     ,
                +       IF      0
                +       ELSE
```

# RELOCATION

The object module produced by this assembler may be in a relocatable format. This allows the user to write programs whose final addresses will be adjusted by Microtec's Linking Loader and which may also be changed without reassembling the complete program. It also allows separate object modules to be linked together into a final program.

Relocatable programming provides many advantages for the user. Actual memory addresses are of no concern until the final load time. Large programs may be easily separated into smaller segments, developed separately, and linked together. If one segment contains an error, only the one need be reassembled. Once developed a library of routines may be used by many users. The Loader will adjust addresses to meet each users requirements.

## Program Segments

To take advantage of relocatability, the user should understand the concept of program segments and how separate object modules are linked together. A program segment is that part of a program which contains its own program counter and is a logically distinct section of the program. At load time the addresses for each segment may be specified separately.

This assembler provides for six program segments. The Program Segment (PSCT) is typically the segment that contains the actual machine instructions. In a RAM/ROM environment it would be the segment that would be placed into ROM. The data area of a program is typically placed into the Base (BSCT) or Data (DSCT) segment. Generally these segments will reside in RAM. The Base Segment represents addresses 0 to 0FFH in the actual microprocessor. By placing variables in the Base Segment, the user may take advantage of 2 byte Zero Page addressing instead of 3 byte Absolute addressing. The Base Segment is the only segment that should have a fixed address at load time. In addition both the assembler and loader will generate errors if the user overflows the Base Segment. The Data Segment is usually used to contain variables which do not fit in the Base Segment or will not reside within addresses 0 to 0FFH at final load time. Both the Base Segment and Data Segment canld contain machine instructions just as the Program Segment can contain data.

The Common Segment (CSCT), like the Data Segment, is used for program variables and resides in RAM. The difference between the Common and Data Segments is in how the Loader processes each of them. Data segments from different programs are placed one after another in memory while Common segments from different programs all begin at the same address. This allows the user to refer to the same location with different names in separate programs. The Common Segment is analagous to COMMON in Fortran and is typically used to link to FORTRAN programs. Another difference

between the Data and Common Segments is that the Common segment may not be initialized, i.e. may not contain instructions at assembly time,although it may reserve storage.

Two additional segments are provided for ease of programming. The Stack Segment may be used to contain the program stack area and resides in RAM. Typically only the main program makes references to the Stack segment and specifies the stack size. References are made to the Stack Segment with the reserved symbol "STACK". The Memory Segment is that portion of memory space not allocated to the other five segments. References are made to this segment with the reserved symbol "MEMORY". Both the Stack and Memory segments are common among all programs and their address and length are formed at load time, not at assembly time. This is in contrast to the Common Segment whose length is determined at assembly time among all programs.

As with non-relocatable assemblers, users may also specify absolute addresses when assembling a program. In this case the object modules, even if in relocatable format, will contain instructions or data which will reside in the specified memory locations.

## Linking

The object modules of the assembler are combined or linked together by a Linking Loader. The Loader converts all relocatable addresses into absolute addresses and resolves references from one module to another. Linkage between modules is provided by external definitions (XDEF,PUBLIC), external references (XREF,XREFB,EXTRN) and the Common Segment. External definitions are defined in one object module and made available to all other object modules via the Linking Loader. External references are symbols referenced in one module but defined in another module. The Linking Loader combines the external definitions from one program with the external references from other programs to obtain the final addresses. A program may contain both external references and definitions.

## Relocatable Symbols

Each symbol in the assembler has associated with it a symbol type which denotes the symbol as absolute or relocatable. If relocatable, it also indicates the segment to which the symbol belongs. Symbols whose values and not dependent upon program origin are absolute symbols. Symbols whose values change when the program origin is changes are called relocatable symbols. The reserved symbols, "STACK" and "MEMORY" discussed above are special forms of a relocatable symbol. External references are also considered relocatable although the final value may be absolute. Absolute and relocatable symbols may both appear in an absolute or relocatable program segment. Note, relocatable symbols may not appear in a program where the "LIST A" directive has been specified.

Absolute symbols are defined as follows:

1. A symbol in the label field of an instruction which is in an absolute segment of code.
2. A symbol is made equal to an absolute expression by the EQU or SET directive. This occurs even if the program is assembling a relocatable segment.

Relocatable symbols are defined as follows:

1. A symbol in the label field of an instruction when the program is assembling a PSCT, BSCT, DSCT, or CSCT segment of code.
2. A symbol is made equal to a relocatable expression by the EQU or SET directives.
3. The reserved symbols STACK and MEMORY are relocatable.
4. External references are relocatable.
5. A reference to the program counter (*) while assembling a relocatable segment is relocatable.

Relocatable symbols are also classified as Program, Data, Base, Common, Stack, or Memory relocatable depending upon how they are defined.

## Relocatable Expressions

The relocatability of an expression is determined by the relocation of the symbols that comprise the expression. All numeric constants are considered absolute. Relocatable expressions may be combined to produce: an absolute expression, a relocatable expression, or in certain instances, illegal expressions. The following list shows those expressions whose result is relocatable: (ABS denotes an absolute symbol, constant, or expression and REL denotes a relocatable symbol or expression)

| ABS+REL | REL+ABS | REL-ABS | .LOW.REL | .HIGH.REL |
|---|---|---|---|---|
| | | | > | < |

In addition, the following expressions are valid and produce an absolute expression. Both relocatable expressions must be relocatable in the same program segment.

| | | |
|---|---|---|
| REL.EQ.REL | REL.NE.REL | REL.LE.REL |
| REL.LT.REL | REL.GE.REL | REL.GT.REL |
| REL-REL | | |

Relocatable symbols that appear in expressions with any other operators will cause an error. e.g. REL*REL. Any combination of two relocatable symbols from different segments, including external references

(XREF,XREFB), is an error.

Zero Page Addressing

As discussed previously, the program will attempt to make an instruction, that can use Zero Page addressing instead of Absolute addressing to save a byte. The user must supply the proper information for the assembler to make the necessary decision. If an operand contains an undefined symbol, the assembler will allocate 3 bytes. Therefore the user should structure the source program so that data variables in the Base Segment or other segments are specified first along with any Common variables. The program which follows would then access variables which will be defined.

The user may also force Zero Page addressing by use of the special character "&". If this character is the first character of an operand that may use either Absolute or Zero Page addressing, the assembler will make it 2 bytes.

The assembler will allocate 2 bytes for an instruction (assuming it may be either 2 or 3 bytes) under the following conditions. (The first 4 conditions assume that the operand is completely defined during Pass 1 of the assembler)

1. An absolute operand has a value between 0-0FFH.
2. An operand that is relocatable in the Base Segment will force the instruction to Zero Page addressing. In this case the Loader will check that the final value does indeed reside in the Base Segment.
3. An external reference declared to be in the Base Segment (XREFB directive) will force the instruction to Zero Page addressing. Once again, the Loader will check that the reference resides in the Base Segment or has a value less than 256.
4. The assembler will force Zero Page addressing if the operand contains the .LOW. or .HIGH. operators. These operators contain only 8 bits of the expression which then allows the instruction to be contained in two bytes. The user may make reference to any part of the program regardless of whether the final reference has a value of 0-0FFH. Obviously an instruction that references an address greater than 0FFH will actually be addressing the Zero Page and cause the program to operate incorrectly. The assembler will not consider this to be an error.

   These operators may also be used when the user wants to use the Data Segment as a separate segment for data (in addition to or instead of the Base Segment) and knows that the segment will reside between 0-0FFH at

load time. This is necessary because any references to the Data Segment, as well as Common and Program Segments, are deemed to reside outside the Zero Page and require 3 bytes for addressing.

5. The assembler will force Zero Page addressing if the first character of the operand is the character "&".

## Relocation Directives

The following pages describe those directives in the assembler that pertain primarily to relocation. The nomenclature is the same as for the directives described in Section 4. The directives are:

| | |
|---|---|
| ASCT | Specify Absolute Segment |
| PSCT | Specify Program Segment |
| DSCT | Specify Data Segment |
| BSCT | Specify Base Segment |
| CSCT | Specify Common Segment |
| ORG | Specify Origin |
| XDEF | Specify External Definition |
| PUBLIC | Same as XDEF |
| XREF | Specify External Reference |
| XREFB | Specify Base Page External Reference |
| EXTRN | Same as XREF |
| NAME | Specify Module Name |
| STKLN | Specify Stack Length |

ASCT - Specify Absolute Segment

The ASCT directive specifies to the Assembler that the following statements should be assembled in the absolute mode. The ASCT remains in effect until a PSCT, BSCT, DSCT, or CSCT directives is assembled. The starting address for the ASCT program counter is zero. At the start of assembly the program assumes an ASCT directive has been specified and assembly proceeds in the absolute mode.

The user may want to use the "LIST A" directive to generate an absolute assembly with an absolute object module.

Note: this directive causes absolute code to be generated within a relocatable program and for which the object module is in relocatable format.

| {label} | ASCT |
|---|---|

where:

label - is an optional label that is assigned the address of the current program counter before the program switches to the absolute mode. It is typically not used.

PSCT - Specify Program Segment

The PSCT directive specifies to the assembler that the following statements should be assembled in the relocatable mode using the Program Segment program counter. This segment remains in effect until another segment directive is assembled. Initially the Program Segment program counter is set to zero.

This directive may specify an operand indicating the relocation type. The relocation type is passed to the Loader and has no effect on the assembly.

The user may alternate between the various segments with multiple segment directives for a given segment within one program. The assembler will maintain the current value of the program counter for each segment.
Example:

```
PSCT    P
```

| {label} | PSCT | {blank,P,I} |
|---|---|---|

where:

label - an optional label which will be assigned the address of the current program counter.

blank - specifies the program segment may be relocated to the next available byte.

P - specifies that the Program Segment must begin on a page boundary (0,100H,200H,...) when relocated by the Linking Loader.

I - specifies that the Program Segment must fit within a single page when relocated. The Loader will start the segment at the next page boundary if the segment will not fit within the current page.

blank represents no operand

If multiple PSCT directives are specified in the same assembly, each must specify the same operand.

DSCT - Specify Data Segment

The DSCT directive specifies that the following statements should be assembled in the relocatable mode using the Data Segment Program Counter. Initially this program counter is set to zero. This segment remains in effect until another segment directive is assembled.

This directive may specify an operand indicating the relocation type. The relocation type is passed to the Loader and has no effect on the assembly.

Unless otherwise denoted, all references to a Data Segment variable assume a 2 byte (as opposed to a 1 byte) address reference.

Example:

```
DSCT   I
```

| {label} | DSCT | {blank,P,I} |
|---|---|---|

where:

label - an optional label which will be assigned the address of the current program counter before the switch is made to the Data Segment.

blank - specifies that the Data Segment may be relocated to the next available byte.

P - specifies that the Data Segment must begin on a page boundary (0,100H,200H,...) when relocated by the Linking Loader.

I - specifies that the Data Segment must fit within a single page when relocated. The Loader will start the segment at the next page boundary if the segment will not fit within the current page.

blank represents no operand

If multiple DSCT directives are specified in the same assembly, each must specify the same operand.

BSCT - Specify Base Segment

The BSCT directive specifies that the following statements should be assembled in the relocatable mode using the Base Segment program counter. This program counter is initially set to zero. This segment remains in effect until another segment directive is assembled.

All references to a variable in the Base Segment will force the appropriate instructions to Zero Page addressing instead of Absolute addressing.

Unlike other segment directives which enable the user to pass paging information to the Loader, the Base segment provides for none. This is the case since this segment must reside in addresses 0-0FFH in the final object module produced by the Loader.

The assembler will issue an error message if the Base Segment in the current assembly exceeds 256 bytes.

Example:

```
BSCT
```

```
{label} | BSCT
```

where:

label - an optional label which will be assigned the address of the current program counter before the switch is made to the Base Segment.

CSCT - Specify Common Segment

The CSCT directive specifies to the assembler that the following statements should be assembled in the relocatable mode using the Common Segment program counter. This segment remains in effect until another segment directive is assembled. Initially this program counter is set to zero.

This directive may specify an operand indicating the relocation type. The relocation type is passed to the Loader and has no effect on the assembly.

Unless otherwise noted, all references to the Common Segment assume a two byte address reference.

No executable code may be placed in the Common Segment therefore, any directives that generate code, or any instructions used in the Common Segment will be flagged with a warning. Typically the RES or *=*+ directive will be used to allocate storage within the Common Segment.

Example:

```
CSCT    P
```

| {label} | CSCT | {blank,P,I} |
|---|---|---|

where:

label - an optional label which will be assigned the address of the current program counter before the switch is made to the Common Segment.

blank - specifies that the Common Segment may be relocated to the next available byte.

P - specifies that the Common Segment must begin on a page boundary (0,100H,200H,...) when relocated by the Linking Loader.

I - specifies that the Common Segment must fit within a single page when relocated. The Loader will start the segment at the next page boundary if the segment will not fit within the current page.

blank represents no operand

If multiple CSCT directives are specified in the same assembly, each must specify the same operand.

ORG - Set Program Origin (relocatable mode)

The ORG directive is used to inform the assembler of the memory address to which the next assembled byte should be assigned. This directive changes the program counter of the segment which is currrently being assembled: Program, Base, Data, or Common. When the ORG is in a relocatable segment, the origin address must be either an absolute expression or a relocatable expression which is relocatable within the current segment.

Example:

```
        ORG     *+30
```

| {label} | ORG | expression |
|---|---|---|
| | *= | expression |

where:

label - an optional label which will be equted to the given expression.

expression - a value which will replace the contents of the current segment program counter. Any symbols used must be previously defined. The expression must be absolute or relocatable in the current segment. No separator need be specified after the *=.

XDEF - Specify External Definition
PUBLIC

These directives specify a list of symbols which will be given the external definition attribute. These symbols will then be made available to other modules by the Linking Loader to establish the necessary linkage between modules. Only those symbols defined by these directives and defined in the assembly are placed into the object module.

These directives may appear anywhere within the program and each symbol may be declared in only one directive.

Example:

```
                        XDEF    SCAN,LABEL,COSINE
```

| | |
|---|---|
| XDEF<br>PUBLIC | symbol list |

where:

symbol list - is a list of symbols separated by commas which specify the names defined in this module and available to other modules.

XREF - Specify External Reference
EXTRN

These directives specify a list of symbols which will be given the external reference attribute. These are symbols that are referenced in this program module but defined within another program. This directive provides the linkage to those symbols through the Linking Loader.

Any symbols specified by these directives are considered to require 2 bytes for an instruction reference. This is the case even if the final symbol is absolute with a value less than 256.

These directives may appear anywhere within the program and each symbol may be declared in only one directive.

Example:

```
                    XREF    INPUT,OUTPUT
```

| | |
|---|---|
| XREF<br>EXTRN | symbol list |

where:

symbol list - is a list of symbols separated by commas which specify the external names available in other modules.

XREFB - Specify Base Page External Reference

This directive is similar to the XREF directive except it implies that any symbols declared by this directive will have a final value less than 256, consequently any instructions referencing this symbol can use Zero Page addressing. The Loader will issue an error message if the symbol is not actually less than 256 in the defining module.

This directive may appear anywhere within the program and each symbol may be declared in only one XREFB directive.

Example:

```
XREFB  DATA
```

| XREFB | symbol list |
|---|---|

where:

symbol list - is a list of symbols separated by commas which specify the external names available in other modules which will reside in the Base Segment or have final values less than 256.

NAME - Specify Module Name

The NAME directive is used to assign a name to the object module produced by the assembler. Only one NAME directive should appear in a program. The module name is a "handle" used by the Linking Loader when combining programs and is the name displayed on the Linking Loader's load map.

If a NAME directive is not specified by the user, the default name "MODULE" is used.

Example:

```
NAME   MULT
```

| | |
|---|---|
| NAME | name |

where:

name - is the name to be placed in the object module denoting the module name to the loader. This name must follow all the rules of a symbol.

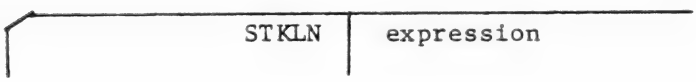
STKLN - Specify Stack Length

The STKLN directive allows the user to specify the length of the Stack Segment generated by the Linking Loader. Typically this directive is only used in the main program, but other programs may also specify a stack length. The Loader combines all Stack Segments into one segment.

If the user does not specify a STKLN directive, the assembler uses a default length of zero. More than one STKLN directive may be placed in a program; only the last one is used.

Example:

```
STKLN  20H
```




where:

expression - an expression which indicates the length of the Stack Segment. This expression may not be relocatable and all symbols used in the expression must be previously defined.

# HOW TO USE THE ASSEMBLER

## The Assembler

The Assembler program is usually supplied as an unlabeled, unblocked magnetic tape with 80 character card image records. Other media may be requested.

The Assembler is written entirely in Fortran and is comprised of a main program and several subroutines. The main program appears first on the tape and the last subroutine is followed by a tape mark. The Assembler may be compiled from tape.

The Assembler Installation Notes describe program installation and any modifications that may have to be made for a particular computer. It is recommended the user read these notes before installing the program.

## Assembler Operation

The Assembler is a two pass program in which the source text is scanned twice. During the first pass the labels are examined and placed into the symbol table. Certain errors may be detected during this pass; these will be displayed on the output listing.

During Pass Two, the object code is generated, symbolic addresses are resolved, and a listing and object module is produced. Errors not detected during Pass One will be detected during Pass Two.

At the end of the Assembly process a symbol table or a cross reference table may be displayed.

The following steps are taken to assemble a program:

1. Write a program utilizing mnemonics and directives. Encode the argument fields with constants, labels, symbolic addresses, etc.

2. Transfer the source program to some computer readable medium; cards, tape, disk file, etc. This medium should correspond to the input device expected by the Assembler. On some systems, device assignments may be changed during the course of an assembly by utilizing proper system control cards.

3. Include the source code as shown in the sequence in Illustration I.

4. Execute the Assembler Program.

5. Get listing and object module as output.

## Assembler Listing

During Pass Two of the assembly process a program listing is produced. The listing displays all information pertaining to the assembled program; both assembled data and the users original source statements.

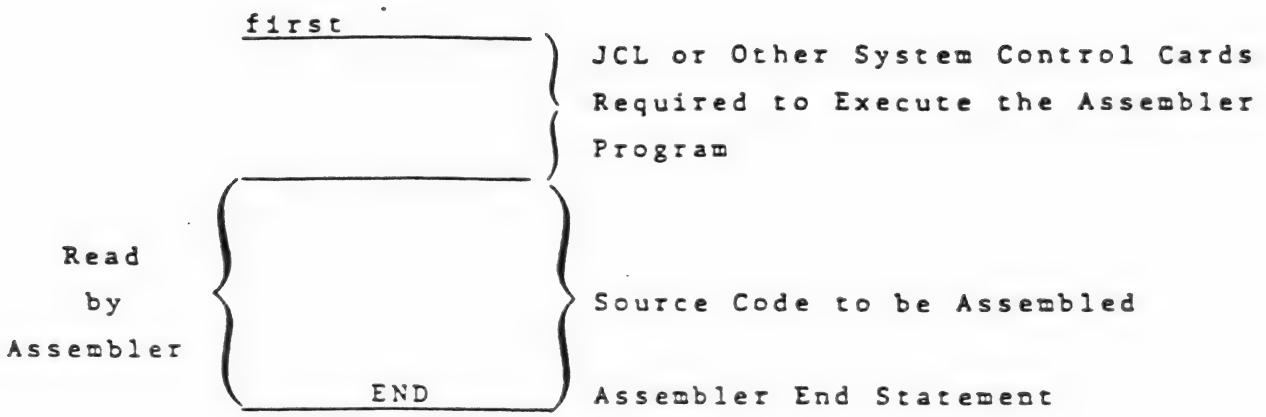
The listing may be used as a documentation tool through the inclusion of comments and remarks that describe the function of the particular program segment.

The main purpose of the listing is to convey all pertinent information about the assembled program, i.e. the memory addresses and their values. The load module, also produced during Pass Two, contains the address and value information but in a format that is not easily read.

CARD ORDER

Illustration I

Read the Input Stream



The illustration on page 7-5 is a sample of a typical program listing. Referring to the illustration, the following information is pertinent:

- When the assembler detects error conditions during the assembly process, the column titled "ERR" will contain error code(s) describing the errors in the associated line of source code. An explanation of the individual error codes is given in Appendix A.

- The column titled "LINE" contains decimal numbers which are associated with the listing source lines. These numbers are used with a cross reference table.

- The column titled "ADDR" contains a value which represents the first memory address of the data shown in bytes one to four on a given line or the value of an EQU or SET directive. The hexadecimal number under B1 represents one byte of data to be stored in the memory address. If there is a number under B2 it represents data to be stored in the given memory addres plus one. Columns B3 and B4, if they contain values, similarly represent data to be stored in the memory address plus two or three.

- To the right of the data bytes are the relocation types of any relocatable operands. The types are: B - Base, C - Common, D - Data, E - External, M - Memory, P - Program, S - Stack.

- The users original source statements are reproduced to the right of the above information. Macro expansions are preceded with a plus sign.

- At the end of the listing the assembler prints the message "ASSEMBLER ERRORS = " with a cumulative count of errors. The assembler substitutes three bytes of zeros when it cannot translate a particular opcode and so provides room for patching the program if desired.

- A symbol table or cross reference table is generated at the end of the assembly listing. The table lists all symbols utilized in alphabetic order along with any relocation types as described above.

```
            1                       *
            2                       * SAMPLE PROGRAM FOR 6500 RELOCATABLE ASSEMBLER
            3                       *
            4                       * INPUT IS FREE FORMAT
            5                       *
            6                       * FOR CERTAIN ERRORS, SPACE IS PROVIDED FOR A PATCH
            7                       *
            8                             LIST    X               LIST CROSS REFERENCES
            9                             NAME    SAMPLE          SET MODULE NAME
           10                             XDEF    STOR1,STOR2     DECLARE DEFINITIONS
           11                             XREF    E1              EXTERNAL REFERENCE
           12                             XREFB   E2              EXTERNAL REFERENCE (BASE PAGE)
           13                             PSCT                    ;SET PROGRAM SEGMENT
           14                      *
           15                      * EXAMPLE OF MACRO CAPABILITY
           16                      *
           17                      MAC1   MACRO   Y,Z
           18                             LDA     #Y
           19                             ADC     #1
           20                             EOR     #255
           21                             JMP     Z
           22                             ENDM
           23                      *
           24                      * EXAMPLE OF VARIOUS ASSEMBLER ERRORS
           25                      *
      0    26 0000 00 00 00       STAR   ADD                     UNDEFINED OPCODE
      V    27 0003 C9 04                 CMP     #260            ILLEGAL VALUE
      U    28 0005 4D 00 00              EOR     U               UNDEFINED SYMBOL
      4    29                            EQU     15              MISSING LABEL
      L    30 0008 00 00 00       AB*C   ASLA                    LABEL ERROR
      S    31 000B 4C 00 00              JMP     STAR++5         SYNTAX ERROR
      F    32 000E 49 16                 EOR     #22,            FORMAT ERROR
      D    33 0010 98             STAR   TYA                     DUPLICATE LABEL
      A    34 0011 A2 00                 LDX     #               ARGUMENT ERROR
      X    35 0013 2E 00 00       B      ROL     SUM,Y           INDEX REGISTER ERROR
      I    36 0016 A1 00                 LDA     (5,Y)           INDIRECT ERROR
      B    37 0018 10 00                 BPL     *+150           BRANCH OUT OF RANGE
      E    38 001A AE 00 00              LDX     SUB#5           RELOCATION ERROR
           39                      * ASSEMBLER DIRECTIVES
           40                            DSCT                    ;SET DATA SEGMENT
           41                            ORG     100             SET ORIGIN
           42 0001                ONE    EQU     1               EQUATE ONE AND 1
           43 0064 0F             STOR1  BYTE    15              DEFINE DATA
           44 0065 36 35 30 30           .BYT    '6500 ASSEMBLER' CHARACTER STRING
           45 0069 20 41 53 53
           46 006D 45 4D 42 4C
           47 0071 45 52
           48 0073 31 00          P      WORD    SUB             DEFINE DOUBLE BYTE
           49                            BSCT                    ;SET BASE SEGMENT
           50 0000                SUM    *=*+5                   RESERVE STORAGE
           51 0005 3F 0D                 STOR2   DATA    3FH,13
           52                            PSCT
           53                      * EXAMPLE OF THE VARIOUS INSTRUCTIONS
           54 0010 86 00          B      BEG1    STX     SUM
```

```
     55 001F EA                          NOP
     56 0020 A9 42                       LDA     #'B             LOAD ASCII CHARACTER
     57 0022 0A                          ASLA
     58 0023 A2 25                       LDX     #$25
     59 0025 E9 1F                       SBC     #1FH
     60 0027 4C 1D 00       P            JMP     BEG
     61 002A 20 22 00       P            JSR     BEG+5
     62 002D 15 00          B            ORA     SUM,X
     63 002F D0 05                       BNE     *+7             LOCATION COUNTER REFERENCE
     64 0031 E8                  SUB     INX
     65 0032 A9 00          C            LDA     #>BUFF
     66 0034 A9 00          B            LDA     #<SUM
     67                          *
     68                          *
     69 0001                     CONTRL  SET     1
     70 0036 78                  MAIN    SEI
     71 0037 BA                          TSX
     72 0038                             MAC1    SUM,MAIN        CALL THE MACRO
     73 0038 A9 00          B    *       LDA     #SUM
     74 003A 69 01                *      ADC     #1
     75 003C 49 FF                *      EOR     #255
     76 003E 4C 36 00       P    *      JMP     MAIN
     77 0041 4E 4F 50 EA                 DATA    'NOP',0EAH
     78                                  NLIST   M               DON'T EXPAND THE MACRO
     79 0045                             MAC1    SUM,STAR
     80 004E 38                          SEC
     81                                  IF      CONTRL=1        CONDITIONAL ASSEMBLY
     82                                  SEI
     83                                  STA     SUM,X
     84                                  ELSE
     85 004F 09 20                       ORA     #45
     86 0051 4C 36 00       P            JMP     MAIN
     87                                  ENDIF
     88                                  IF      CONTRL
     89 0054 78                          SEI
     90 0055 95 02          B            STA     SUM+2,X
     91 0057 09 20                       ORA     #45
     92                                  ELSE
     93                                  ORA     #0FFH
     94                                  JMP     MAIN
     95                                  ENDIF
     96                          *
     97                                  CSCT    P               SET COMMON SEGMENT
     98 0000                     BUFF    RES     10
     99 000A                             END
```

ASSEMBLER ERRORS = 13

CROSS REFERENCE

| LABEL | | VALUE | REFERENCE | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| BEG | P | 0010 | -54 | 60 | 61 | | | | | |
| BUFF | C | 0000 | 65 | -102 | | | | | | |
| CONTRL | | 0001 | -69 | 85 | 92 | | | | | |
| E1 | E | 0000 | 11 | | | | | | | |
| E2 | E | 0001 | 12 | | | | | | | |
| MAIN | P | 0036 | -70 | 76 | 90 | | | | | |
| MEMORY | M | 0000 | 0 | | | | | | | |
| NARG | | 0000 | 0 | | | | | | | |
| ONE | | 0001 | -42 | | | | | | | |
| STACK | S | 0000 | 0 | | | | | | | |
| STAR | P | 0000 | -26 | 31 | -33 | 83 | | | | |
| STOR1 | D | 0064 | 10 | -43 | | | | | | |
| STOR2 | B | 0005 | 10 | -51 | | | | | | |
| SUB | P | 0031 | 38 | 48 | -64 | | | | | |
| SUM | B | 0000 | 35 | -50 | 54 | 62 | 66 | 73 | 80 | 94 |

```
023E0006SAMPLE00300159000302750003030A0002040700030500000306000000337
18160006E1****0006E2****01A2
16120002640006STOR1*00FA
16120004050006STOR2*0056
063C0001000000000C9044D00000000004C0000491698A2002E0000A10010000F
240A000403140087
060E00011A00AE000023
062A000264000F3635303020415353454D424C45523100A1
240A000103730058
060C000405003F0099
06320001ID008600EAA9420AA225E91F4C1D00202200150000005E8F9
220C00032800280070
240E0004031E002E007B
060C00013200A90012
240A000301330098
06100001340OA90078BADA
240C00040235000095
06400001380OA900690149FF4C36004E4F50EAA900690149FF4C00003809204C360036
221000033F004C005200EE
240E0004033900460048
06120001540078950209204E
240A000403560075
040A00000100000F1
0E0200F0
```

## The Object Module

As part of the Pass Two processing, the assembler produces an object module. The object module is a machine readable computer output in the form of punched cards, paper tape, etc. The output module contains specifications for loading the memory of the target microprocessor and provides the necessary linkage to link object modules together. If the user has specified an absolute assembly with the "LIST A" directive, an absolute object module is produced.

The object module is normally punched out on the device specified as a series of card image records.

The object module may be loaded into Microtec's Linking Loader which will then convert it to an absolute program in MOS Technology's format. This is compatible with the KIM and TIM hexadecimal format The absolute object module may then be loaded into a development system or used to program a PROM.

A sample relocatable object module is shown on the preceding page. This is the object module of the sample program.

The format of the absolute object module is described below. This module consists of symbol table information, data specifications for loading memory, as well as a module terminator record.

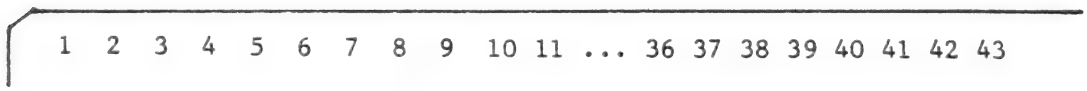
```
$$
 symbol records
$$
 data records
 terminator record
```

## Symbol Record

As many symbol records as needed may be contained in the object module. At most 4 symbols per line are used but each line need not contain 4 symbols even if it is not the last line. A module may contain no symbol records. A sample record is shown below.

```
APPLE   00000H   LABEL1  0D0C3H   MEM     0FFFFH   ZEEK    01947H
```

Data Record




Column 1 contains a semi-colon. This marks the beginning of a data record.

Columns 2 and 3 contain the count of the number of data bytes on the record. If this field contains a "00" it signifies the end of the object module.

Columns 4 through 7 contain the load address. The first data byte is to be loaded into this address, subsequent bytes into the next sequential addresses. Columns 4 and 5 contain the high order address byte.

Columns 8 through 39 (less if not 16 data bytes) contain the specifications for up to 16 bytes of data.

The last 4 columns in the data record contain a checksum. The checksum is the sum, to 16 bits, of all bytes on the record (except column 1)

Terminator Record

A Terminator Record has the same format as a Data Record with the data byte count (columns 2 and 3) set to zero and the load address field (columns 4 through 7) containing the count of the number of Data Records.

## Cross Reference Format

The cross reference option is normally turned off. To turn it on use "LIST X", to turn it off use "NLIST X" (see LIST and NLIST directives). The assembler will produce either a cross reference table or a symbol table. The cross reference table will be produced if "LIST X" has been specified. References may be accumulated only during selected portions of the program by turning the cross reference option on and off. However, to obtain the cross reference listing, the option must be turned on before the END directive. Typically the "LIST X" directive will be one of the first statements in the source program and never turned off.

An example of the cross reference output is shown on the sample listing earlier in this section. On this listing, LABEL and VALUE are self explanatory. Any flags on the left of the value are the relocation types of the symbols. Under REFERENCE, a value preceded by a minus sign indicates that the symbol was defined on that line. A value of 0 as the only entry on the line indicates this is an internal system symbol (e.g. STACK). Line numbers not preceded by a minus sign indicate a reference to a symbol. Note that for SET symbols or for multiply defined symbols, more than one definition may appear for the symbol, as for the symbol STAR on the listing.

APPENDIX A

## ASSEMBLER ERROR CODES

If errors in the source program are detected during the assembly process, an indication of the type of error is printed on the listing on the same line as the statement in error.

The following list should serve as a guide to diagnose the error. The listing always displays a total error count.

A - Argument error. The arguement is missing or contains an illegal character, etc.

B - Branch error. A branch instruction is attempting to branch to a location which is out of range for the relative address. This includes attempts to branch past either end of the address space (0 or 0FFFFH).

C - Macro Substitution error. When substituting actual macro parameters for formal macro parameters, the 80 column source line limit was exceeded.

D - Duplicate Label. The label in the statement has previously appeared in the label field. A label on a SET directive previously appered in a statement other than a SET directive or a label on a statement other than a SET directive now appears on a SET directive. A label appears more than once in a XREF, XREFB, or XDEF directive. A symbol defined in a XREF or XREFB directive appears in the label field of some statement.

E - Relocation error. The instruction contains an operand that violates a rule of relocation. An operand that should be absolute is relocatable or an EQU, SET, or ORG directive makes reference to an external symbol. A relocatable expression appears in an absolute only assembly. A relative branch instruction is attempting to branch to an address in another segment.

F - Format error. The instruction has been written in a format which is not permitted. This error usually indicates a trailing comma and the instruction is assembled properly.

I - Indirect error. X or Y not specified for indirect addressing. X indexing specified for format that requires Y indexing or vice-versa. Closing parenthesis not found for indexed indirect instruction.

L - Label error. A label contains an invalid character or starts with a numeric character.

M - Missing Label. This statement requires a label.

N - Macro Nesting error. When nesting macros the buffer available for macro parameters is full.

O - Opcode error. The opcode mnemonic is not a valid instruction, directive, or a macro call. Also a macro defined within another macro, or conditional assembly statements nested too deeply. ELSE, ENDIF, or ENDM used without preceding IF or MACRO. LOCAL directive used outside of MACRO body or more than one NAME directive in program.

S - Syntax error. A rule of syntax has been violated in this statement. Parentheses are not nested properly or possibly two operators appear in sequence, e.g. +>5.

T - Table Overflow error. Symbol table is full - assembly continues. An attempt was made to define too many macros. Program counter advanced to a value larger than 65535. In this case the value is made modulo 65536. The Base Page exceeds a value of 255.

U - Undefined symbol. There is a symbolic name in the operand field which has never been defined. The symbol should have been previously defined for certain directives and was not, but may have been defined after the directive. A symbol declared on the XDEF directive was not used in the program.

V - Value error. An evaluated expression or constant is out of range for the field of the actual machine instruction for which it is to be contained. A one byte value is relocatable but was not preceded by a .LOW. or .HIGH. operator or is not in the Base Segment. In this case it is forced to .LOW.

W - Warning. This instruction could have been assembled with Zero Page addressing. A relocatable directive has been specified during an absolute assembly or an attempt was made to initialize the CSCT segment. In the last case, the statement is ignored

X - Indexing error. A symbol other than X or Y was specified for indexing. Indexing expected but no symbol found. X indexing specified for instruction that uses Y index register or vice-versa.

CROSS REFERENCE OVERFLOW AT ____. The cross reference file has been filled. Assembly continues and references stop. Enlarge cross reference table or file. (See Installation Notes)

## APPENDIX B

## ASCII AND EBCDIC CODES

The Assembler will recognize only the following characters. The equivalent codes are expressed in hexadecimal notation.

| CHARACTER | ASCII | EBCDIC |
|---|---|---|
| 0 | 30 | F0 |
| 1 | 31 | F1 |
| 2 | 32 | F2 |
| 3 | 33 | F3 |
| 4 | 34 | F4 |
| 5 | 35 | F5 |
| 6 | 36 | F6 |
| 7 | 37 | F7 |
| 8 | 38 | F8 |
| 9 | 39 | F9 |
| A | 41 | C1 |
| B | 42 | C2 |
| C | 43 | C3 |
| D | 44 | C4 |
| E | 45 | C5 |
| F | 46 | C6 |
| G | 47 | C7 |
| H | 48 | C8 |
| I | 49 | C9 |
| J | 4A | D1 |
| K | 4B | D2 |
| L | 4C | D3 |
| M | 4D | D4 |
| N | 4E | D5 |
| O | 4F | D6 |
| P | 50 | D7 |
| Q | 51 | D8 |
| R | 52 | D9 |
| S | 53 | E2 |
| T | 54 | E3 |
| U | 55 | E4 |
| V | 56 | E5 |
| W | 57 | E6 |
| X | 58 | E7 |
| Y | 59 | E8 |
| Z | 5A | E9 |
| blank | 20 | 40 |
| ! | 21 | 5A |
| " | 22 | 7F |
| # | 23 | 7B |
| $ | 24 | 5B |
| % | 25 | 6C |
| & | 26 | 50 |
| ' | 27 | 7D |
| ( | 28 | 4D |
| ) | 29 | 5D |
| * | 2A | 5C |
| + | 2B | 4F |
| , | 2C | 6B |
| - | 2D | 60 |
| . | 2E | 4B |
| / | 2F | 61 |
| : | 3A | 7A |
| ; | 3B | 5E |
| < | 3C | 4C |
| = | 3D | 7E |
| > | 3E | 6E |
| ? | 3F | 6F |
| @ | 40 | 7C |
| _ | 5F | 6D |

APPENDIX C

## 6500 OPERATION CODES

The following table illustrates the proper format for writing 6500 instructions. The operation codes mnemonics listed are the only valid opcodes for the assembler.

These symbols are used in the table.

```
  X - indexed addressing using X index register
  Y - indexed addressing using Y index register
  Z - Zero Page addressing
 aa - operand of a relative branch instruction
ABS - Absolute addressing
  # - immediate addressing
```

The various addressing modes and an example of each is shown below.

| | | | |
|---|---|---|---|
| Accumulator | ASLA | or ASL A | |
| Immediate | ADC | #exp | |
| Zero Page | ADC | Z | (Z < 256) |
| Zero Page Indexed | ADC | Z,X | |
| | ADC | Z,Y | |
| Absolute | ADC | ABS | |
| Absolute Indexed | ADC | ABS,X | |
| | ADC | ABS,Y | |
| Implied | CLV | | |
| Relative | BEQ | aa | |
| Indexed Indirect | ADC | (Z,X) | |
| Indirect Indexed | ADC | (Z),Y | |
| Absolute Indirect | JMP | (ABS) | |

The following notes apply to the table.

```
 * - add one cycle if indexing across page boundary
** - add one cycle if branch is taken and one additional
     cycle if branch crosses page boundary
```

| SYMBOLIC OPCODE | | FIRST BYTE MACHINE CODE | NUMBER OF BYTES | NUMBER OF CYCLES |
|---|---|---|---|---|
| ADC | # | 69 | 2 | 2 |
| ADC | Z | 65 | 2 | 3 |
| ADC | Z,X | 75 | 2 | 4 |
| ADC | ABS | 6D | 3 | 4 |
| ADC | ABS,X | 70 | 3 | 4* |
| ADC | ABS,Y | 79 | 3 | 4* |
| ADC | (Z,X) | 61 | 2 | 6 |
| ADC | (Z),Y | 71 | 2 | 5* |
| AND | # | 29 | 2 | 2 |
| AND | Z | 25 | 2 | 3 |
| AND | Z,X | 35 | 2 | 4 |
| AND | ABS | 2D | 3 | 4 |
| AND | ABS,X | 3D | 3 | 4* |
| AND | ABS,Y | 39 | 3 | 4 |
| AND | (Z,X) | 21 | 2 | 6 |
| AND | (Z),Y | 31 | 2 | 5* |
| ASL | A | 0A | 1 | 2 |
| ASL | Z | 06 | 2 | 5 |
| ASL | Z,X | 16 | 2 | 6 |
| ASL | ABS | 0E | 3 | 6 |
| ASL | ABS,X | 1E | 3 | 7 |
| BCC | aa | 90 | 2 | 2** |
| BCS | aa | B0 | 2 | 2** |
| BEQ | aa | F0 | 2 | 2** |
| BIT | Z | 24 | 2 | 3 |
| BIT | ABS | 2C | 3 | 4 |
| BMI | aa | 30 | 2 | 2** |
| BNE | aa | D0 | 2 | 2** |
| BPL | aa | 10 | 2 | 2** |
| BRK | | 00 | 1 | 7 |
| BVC | aa | 50 | 2 | 2** |
| BVS | aa | 70 | 2 | 2** |
| CLC | | 18 | 1 | 2 |
| CLD | | D8 | 1 | 2 |
| CLI | | 58 | 1 | 2 |
| CLV | | B8 | 1 | 2 |
| CMP | # | C9 | 2 | 2 |
| CMP | Z | C5 | 2 | 3 |
| CMP | Z,X | D5 | 2 | 4 |
| CMP | ABS | CD | 3 | 4 |
| CMP | ABS,X | DD | 3 | 4* |
| CMP | ABS,Y | D9 | 3 | 4* |
| CMP | (Z,X) | C1 | 2 | 6 |
| CMP | (Z),Y | D1 | 2 | 5* |

| SYMBOLIC OPCODE | | FIRST BYTE MACHINE CODE | NUMBER OF BYTES | NUMBER OF CYCLES |
|---|---|---|---|---|
| CPX | # | E0 | 2 | 2 |
| CPX | Z | E4 | 2 | 3 |
| CPX | ABS | EC | 3 | 4 |
| CPY | # | C0 | 2 | 2 |
| CPY | Z | C4 | 2 | 3 |
| CPY | ABS | CC | 3 | 4 |
| DEC | Z | C6 | 2 | 5 |
| DEC | Z,X | D6 | 2 | 6 |
| DEC | ABS | CE | 3 | 6 |
| DEC | ABS,X | DE | 3 | 7 |
| DEX | | CA | 1 | 2 |
| DEY | | 88 | 1 | 2 |
| EOR | # | 49 | 2 | 2 |
| EOR | Z | 45 | 2 | 3 |
| EOR | Z,X | 55 | 2 | 4 |
| EOR | ABS | 4D | 3 | 4 |
| EOR | ABS,X | 5D | 3 | 4* |
| EOR | ABS,Y | 59 | 3 | 4* |
| EOR | (Z,X) | 41 | 2 | 6 |
| EOR | (Z),Y | 51 | 2 | 5* |
| INC | Z | E6 | 2 | 5 |
| INC | Z,X | F6 | 2 | 6 |
| INC | ABS | EE | 3 | 7 |
| INC | ABS,X | FE | 3 | 7 |
| INX | | E8 | 1 | 2 |
| INY | | C8 | 1 | 2 |
| JMP | ABS | 4C | 3 | 3 |
| JMP | (ABS) | 6C | 3 | 5 |
| JSR | ABS | 20 | 3 | 6 |
| LDA | # | A9 | 2 | 2 |
| LDA | Z | A5 | 2 | 3 |
| LDA | Z,X | B5 | 2 | 4 |
| LDA | ABS | AD | 3 | 4 |
| LDA | ABS,X | BD | 3 | 4* |
| LDA | ABS,Y | B9 | 3 | 4* |
| LDA | (Z,X) | A1 | 2 | 6 |
| LDA | (Z),Y | B1 | 2 | 5* |
| LDX | # | A2 | 2 | 2 |
| LDX | Z | A6 | 2 | 3 |
| LDX | Z,Y | B6 | 2 | 4 |

| SYMBOLIC OPCODE | | FIRST BYTE MACHINE CODE | NUMBER OF BYTES | NUMBER OF CYCLES |
|---|---|---|---|---|
| LDX | ABS | AE | 3 | 4 |
| LDX | ABS,Y | BE | 3 | 4* |
| LDY | # | A | 2 | 2 |
| LDY | Z,X | B4 | 2 | 4 |
| LDY | ABS | AC | 3 | 4 |
| LDY | ABS,X | BC | 3 | 4* |
| LSR | A | 4A | 1 | 2 |
| LSR | Z | 46 | 2 | 5 |
| LSR | Z,X | 56 | 2 | 6 |
| LSR | ABS | 4E | 3 | 6 |
| LSR | ABS,X | 5E | 3 | 7 |
| NOP | | EA | 1 | 2 |
| ORA | # | 09 | 2 | 2 |
| ORA | Z | 05 | 2 | 3 |
| ORA | Z,X | 15 | 2 | 4 |
| ORA | ABS | 0D | 3 | 4 |
| ORA | ABS,X | 1D | 3 | 4* |
| ORA | ABS,Y | 19 | 3 | 4* |
| ORA | (Z,X) | 01 | 2 | 6 |
| ORA | (Z),Y | 11 | 2 | 5* |
| PHA | | 48 | 1 | 3 |
| PHP | | 08 | 1 | 3 |
| PLA | | 68 | 1 | 4 |
| PLP | | 28 | 1 | 4 |
| ROL | A | 2A | 1 | 2 |
| ROL | Z | 26 | 2 | 5 |
| ROL | Z,X | 36 | 2 | 6 |
| ROL | ABS | 2E | 3 | 6 |
| ROL | ABS,X | 3E | 3 | 7 |
| ROR | A | 6A | 1 | 2 |
| ROR | Z | 66 | 2 | 5 |
| ROR | Z,X | 76 | 2 | 6 |
| ROR | ABS | 6E | 3 | 6 |
| ROR | ABS,X | 7E | 3 | 7 |
| RTI | | 40 | 1 | 6 |
| RTS | | 60 | 1 | 6 |
| SBC | # | A9 | 2 | 2 |
| SBC | Z | A5 | 2 | 3 |
| SBC | Z,X | B5 | 2 | 4 |
| SBC | ABS | AD | 3 | 4 |

| SYMBOLIC OPCODE | | FIRST BYTE MACHINE CODE | NUMBER OF BYTES | NUMBER OF CYCLES |
|---|---|---|---|---|
| SBC | ABS,X | BD | 3 | 4* |
| SBC | ABS,Y | B9 | 3 | 4* |
| SBC | (Z,X) | A1 | 2 | 6 |
| SBC | (Z),Y | B1 | 2 | 5* |
| SED | | F8 | 1 | 2 |
| SEI | | 78 | 1 | 2 |
| STA | Z | 85 | 2 | 3 |
| STA | Z,X | 95 | 2 | 4 |
| STA | ABS | 8D | 3 | 4 |
| STA | ABS,X | 90 | 3 | 5 |
| STA | ABS,Y | 99 | 3 | 5 |
| STA | (Z,X) | 81 | 2 | 6 |
| STA | (Z),Y | 91 | 2 | 6 |
| STX | Z | 86 | 2 | 3 |
| STX | Z,Y | 96 | 2 | 4 |
| STX | ABS | 8E | 3 | 4 |
| STY | Z | 84 | 2 | 3 |
| STY | Z,X | 94 | 2 | 4 |
| STY | ABS | 8C | 3 | 4 |
| TAX | | AA | 1 | 2 |
| TAY | | A8 | 1 | 2 |
| TYA | | 98 | 1 | 2 |
| TSX | | BA | 1 | 2 |
| TXA | | 8A | 1 | 2 |
| TXS | | 9A | 1 | 2 |

## APPENDIX D

## HEXADECIMAL NOTATION

Hexadecimal notation is a convenient way to express binary information. Each hexadecimal digit may be though of as representing the information in four binary bits.

The assembled code is expressed in hexadecimal notation on the output listing. Hexadecimal is the name of the base 16 number system.

| DECIMAL | HEXADECIMAL | BINARY |
|---|---|---|
| 0 | 0 | 0000 |
| 1 | 1 | 0001 |
| 2 | 2 | 0010 |
| 3 | 3 | 0011 |
| 4 | 4 | 0100 |
| 5 | 5 | 0101 |
| 6 | 6 | 0110 |
| 7 | 7 | 0111 |
| 8 | 8 | 1000 |
| 9 | 9 | 1001 |
| 10 | A | 1010 |
| 11 | B | 1011 |
| 12 | C | 1100 |
| 13 | D | 1101 |
| 14 | E | 1110 |
| 15 | F | 1111 |

## APPENDIX E

## HEXADECIMAL-DECIMAL CONVERSION TABLE

This table allows conversions to be made between hexadecimal and decimal numbers. The table has a decimal range of 0 to 4095. To convert larger numbers add the following values to the table values.

| HEXADECIMAL | DECIMAL |
|---|---|
| 1000 | 4096 |
| 2000 | 8192 |
| 3000 | 12228 |
| 4000 | 16384 |
| 5000 | 20480 |
| 6000 | 24576 |
| 7000 | 28672 |
| 8000 | 32768 |
| 9000 | 36864 |
| A000 | 40960 |
| B000 | 45056 |
| C000 | 49152 |
| D000 | 53248 |
| E000 | 57344 |
| F000 | 61440 |
| 10000 | 65536 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 000 | 0000 | 0001 | 0002 | 0003 | 0004 | 0005 | 0006 | 0007 | 0008 | 0009 | 0010 | 0011 | 0012 | 0013 | 0014 | 0015 |
| 010 | 0016 | 0017 | 0018 | 0019 | 0020 | 0021 | 0022 | 0023 | 0024 | 0025 | 0026 | 0027 | 0028 | 0029 | 0030 | 0031 |
| 020 | 0032 | 0033 | 0034 | 0035 | 0036 | 0037 | 0038 | 0039 | 0040 | 0041 | 0042 | 0043 | 0044 | 0045 | 0046 | 0047 |
| 030 | 0048 | 0049 | 0050 | 0051 | 0052 | 0053 | 0054 | 0055 | 0056 | 0057 | 0058 | 0059 | 0060 | 0061 | 0062 | 0063 |
| 040 | 0064 | 0065 | 0066 | 0067 | 0068 | 0069 | 0070 | 0071 | 0072 | 0073 | 0074 | 0075 | 0076 | 0077 | 0078 | 0079 |
| 050 | 0080 | 0081 | 0082 | 0083 | 0084 | 0085 | 0086 | 0087 | 0088 | 0089 | 0090 | 0091 | 0092 | 0093 | 0094 | 0095 |
| 060 | 0096 | 0097 | 0098 | 0099 | 0100 | 0101 | 0102 | 0103 | 0104 | 0105 | 0106 | 0107 | 0108 | 0109 | 0110 | 0111 |
| 070 | 0112 | 0113 | 0114 | 0115 | 0116 | 0117 | 0118 | 0119 | 0120 | 0121 | 0122 | 0123 | 0124 | 0125 | 0126 | 0127 |
| 080 | 0128 | 0129 | 0130 | 0131 | 0132 | 0133 | 0134 | 0135 | 0136 | 0137 | 0138 | 0139 | 0140 | 0141 | 0142 | 0143 |
| 090 | 0144 | 0145 | 0146 | 0147 | 0148 | 0149 | 0150 | 0151 | 0152 | 0153 | 0154 | 0155 | 0156 | 0157 | 0158 | 0159 |
| 0A0 | 0160 | 0161 | 0162 | 0163 | 0164 | 0165 | 0166 | 0167 | 0168 | 0169 | 0170 | 0171 | 0172 | 0173 | 0174 | 0175 |
| 0B0 | 0176 | 0177 | 0178 | 0179 | 0180 | 0181 | 0182 | 0183 | 0184 | 0185 | 0186 | 0187 | 0188 | 0189 | 0190 | 0191 |
| 0C0 | 0192 | 0193 | 0194 | 0195 | 0196 | 0197 | 0198 | 0199 | 0200 | 0201 | 0202 | 0203 | 0204 | 0205 | 0206 | 0207 |
| 0D0 | 0208 | 0209 | 0210 | 0211 | 0212 | 0213 | 0214 | 0215 | 0216 | 0217 | 0218 | 0219 | 0220 | 0221 | 0222 | 0223 |
| 0E0 | 0224 | 0225 | 0226 | 0227 | 0228 | 0229 | 0230 | 0231 | 0232 | 0233 | 0234 | 0235 | 0236 | 0237 | 0238 | 0239 |
| 0F0 | 0240 | 0241 | 0242 | 0243 | 0244 | 0245 | 0246 | 0247 | 0248 | 0249 | 0250 | 0251 | 0252 | 0253 | 0254 | 0255 |

## HEXADECIMAL-DECIMAL INTEGER CONVERSION (Cont'd)

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 100 | 0256 | 0257 | 0258 | 0259 | 0260 | 0261 | 0262 | 0263 | 0264 | 0265 | 0266 | 0267 | 0268 | 0269 | 0270 | 0271 |
| 110 | 0272 | 0273 | 0274 | 0275 | 0276 | 0277 | 0278 | 0279 | 0280 | 0281 | 0282 | 0283 | 0284 | 0285 | 0286 | 0287 |
| 120 | 0288 | 0289 | 0290 | 0291 | 0292 | 0293 | 0294 | 0295 | 0296 | 0297 | 0298 | 0299 | 0300 | 0301 | 0302 | 0303 |
| 130 | 0304 | 0305 | 0306 | 0307 | 0308 | 0309 | 0310 | 0311 | 0312 | 0313 | 0314 | 0315 | 0316 | 0317 | 0318 | 0319 |
| 140 | 0320 | 0321 | 0322 | 0323 | 0324 | 0325 | 0326 | 0327 | 0328 | 0329 | 0330 | 0331 | 0331 | 0333 | 0334 | 0335 |
| 150 | 0336 | 0337 | 0338 | 0339 | 0340 | 0341 | 0342 | 0343 | 0344 | 0345 | 0346 | 0347 | 0348 | 0349 | 0350 | 0351 |
| 160 | 0352 | 0353 | 0354 | 0355 | 0356 | 0357 | 0358 | 0359 | 0360 | 0361 | 0362 | 0363 | 0364 | 0365 | 0366 | 0367 |
| 170 | 0368 | 0369 | 0370 | 0371 | 0372 | 0373 | 0374 | 0375 | 0376 | 0377 | 0378 | 0379 | 0380 | 0381 | 0382 | 0383 |
| 180 | 0384 | 0385 | 0386 | 0387 | 0388 | 0389 | 0390 | 0391 | 0392 | 0393 | 0394 | 0395 | 0396 | 0397 | 0398 | 0399 |
| 190 | 0400 | 0401 | 0402 | 0403 | 0404 | 0405 | 0406 | 0407 | 0408 | 0409 | 0410 | 0411 | 0412 | 0413 | 0414 | 0415 |
| 1A0 | 0416 | 0417 | 0418 | 0419 | 0420 | 0421 | 0422 | 0423 | 0424 | 0425 | 0426 | 0427 | 0428 | 0429 | 0430 | 0431 |
| 1B0 | 0432 | 0433 | 0434 | 0435 | 0436 | 0437 | 0438 | 0439 | 0440 | 0441 | 0442 | 0443 | 0444 | 0445 | 0446 | 0447 |
| 1C0 | 0448 | 0449 | 0450 | 0451 | 0452 | 0453 | 0454 | 0455 | 0456 | 0457 | 0458 | 0459 | 0460 | 0461 | 0462 | 0463 |
| 1D0 | 0464 | 0465 | 0466 | 0467 | 0468 | 0469 | 0470 | 0471 | 0472 | 0473 | 0474 | 0475 | 0476 | 0477 | 0478 | 0479 |
| 1E0 | 0480 | 0481 | 0482 | 0483 | 0484 | 0485 | 0486 | 0487 | 0488 | 0489 | 0490 | 0491 | 0492 | 0493 | 0494 | 0495 |
| 1F0 | 0496 | 0497 | 0498 | 0499 | 0500 | 0501 | 0502 | 0503 | 0504 | 0505 | 0506 | 0507 | 0508 | 0509 | 0510 | 0511 |
| 200 | 0512 | 0513 | 0514 | 0515 | 0516 | 0517 | 0518 | 0519 | 0520 | 0521 | 0522 | 0523 | 0524 | 0525 | 0526 | 0527 |
| 210 | 0528 | 0529 | 0530 | 0531 | 0532 | 0533 | 0534 | 0535 | 0536 | 0537 | 0538 | 0539 | 0540 | 0541 | 0542 | 0543 |
| 220 | 0544 | 0545 | 0546 | 0547 | 0548 | 0549 | 0550 | 0551 | 0552 | 0553 | 0554 | 0555 | 0556 | 0557 | 0558 | 0559 |
| 230 | 0560 | 0561 | 0562 | 0563 | 0564 | 0565 | 0566 | 0567 | 0568 | 0569 | 0570 | 0571 | 0572 | 0573 | 0574 | 0575 |
| 240 | 0576 | 0577 | 0578 | 0579 | 0580 | 0581 | 0582 | 0583 | 0584 | 0585 | 0586 | 0587 | 0588 | 0589 | 0590 | 0591 |
| 250 | 0592 | 0593 | 0594 | 0595 | 0596 | 0597 | 0598 | 0599 | 0600 | 0601 | 0602 | 0603 | 0604 | 0605 | 0606 | 0607 |
| 260 | 0608 | 0609 | 0610 | 0611 | 0612 | 0613 | 0614 | 0615 | 0616 | 0617 | 0618 | 0619 | 0620 | 0621 | 0622 | 0623 |
| 270 | 0624 | 0625 | 0626 | 0627 | 0628 | 0629 | 0630 | 0631 | 0632 | 0633 | 0634 | 0635 | 0636 | 0637 | 0638 | 0639 |
| 280 | 0640 | 0641 | 0642 | 0643 | 0644 | 0645 | 0646 | 0647 | 0648 | 0649 | 0650 | 0651 | 0652 | 0653 | 0654 | 0655 |
| 290 | 0656 | 0657 | 0658 | 0659 | 0660 | 0661 | 0662 | 0663 | 0664 | 0665 | 0666 | 0667 | 0668 | 0669 | 0670 | 0671 |
| 2A0 | 0672 | 0673 | 0674 | 0675 | 0676 | 0677 | 0678 | 0679 | 0680 | 0681 | 0682 | 0683 | 0684 | 0685 | 0686 | 0687 |
| 2B0 | 0688 | 0689 | 0690 | 0691 | 0692 | 0693 | 0694 | 0695 | 0696 | 0697 | 0698 | 0699 | 0700 | 0701 | 0702 | 0703 |
| 2C0 | 0704 | 0705 | 0706 | 0707 | 0708 | 0709 | 0710 | 0711 | 0712 | 0713 | 0714 | 0715 | 0716 | 0717 | 0718 | 0719 |
| 2D0 | 0720 | 0721 | 0722 | 0723 | 0724 | 0725 | 0726 | 0727 | 0728 | 0729 | 0730 | 0731 | 0732 | 0733 | 0734 | 0735 |
| 2E0 | 0736 | 0737 | 0738 | 0739 | 0740 | 0741 | 0742 | 0743 | 0744 | 0745 | 0746 | 0747 | 0748 | 0749 | 0750 | 0751 |
| 2F0 | 0752 | 0753 | 0754 | 0755 | 0756 | 0757 | 0758 | 0759 | 0760 | 0761 | 0762 | 0763 | 0764 | 0765 | 0766 | 0767 |
| 300 | 0768 | 0769 | 0770 | 0771 | 0772 | 0773 | 0774 | 0775 | 0776 | 0777 | 0778 | 0779 | 0780 | 0781 | 0782 | 0783 |
| 310 | 0784 | 0785 | 0786 | 0787 | 0788 | 0789 | 0790 | 0791 | 0792 | 0793 | 0794 | 0795 | 0796 | 0797 | 0798 | 0799 |
| 320 | 0800 | 0301 | 0802 | 0803 | 0804 | 0805 | 0806 | 0807 | 0808 | 0809 | 0810 | 0811 | 0812 | 0813 | 0814 | 0815 |
| 330 | 0816 | 0817 | 0818 | 0819 | 0820 | 0821 | 0822 | 0823 | 0824 | 0825 | 0826 | 0827 | 0828 | 0829 | 0830 | 0831 |
| 340 | 0832 | 0833 | 0834 | 0835 | 0836 | 0837 | 0838 | 0639 | 0840 | 0841 | 0842 | 0843 | 0844 | 0845 | 0846 | 0847 |
| 350 | 0848 | 0849 | 0850 | 0851 | 0852 | 0853 | 0854 | 0855 | 0856 | 0857 | 0858 | 0859 | 0860 | 0861 | 0862 | 0863 |
| 360 | 0864 | 0865 | 0866 | 0867 | 0868 | 0869 | 0870 | 0871 | 0872 | 0873 | 0874 | 0875 | 0876 | 0877 | 0878 | 0879 |
| 370 | 0880 | 0881 | 0882 | 0883 | 0884 | 0885 | 0886 | 0887 | 0888 | 0889 | 0890 | 0891 | 0892 | 0893 | 0894 | 0895 |
| 380 | 0896 | 0897 | 0898 | 0899 | 0900 | 0901 | 0902 | 0903 | 0904 | 0905 | 0906 | 0907 | 0908 | 0909 | 0910 | 0911 |
| 390 | 0212 | 0913 | 0914 | 0915 | 0916 | 0917 | 0918 | 0919 | 0920 | 0921 | 0922 | 0923 | 0924 | 0925 | 0926 | 0927 |
| 3A0 | 0928 | 0929 | 0930 | 0931 | 0932 | 0933 | 0934 | 0935 | 0936 | 0937 | 0938 | 0939 | 0940 | 0941 | 0942 | 0943 |
| 3B0 | 0944 | 0945 | 0946 | 0947 | 0948 | 0949 | 0950 | 0951 | 0952 | 0953 | 0954 | 0955 | 0956 | 0957 | 0958 | 0959 |
| 3C0 | 0960 | 0961 | 0962 | 0963 | 0964 | 0965 | 0966 | 0967 | 0968 | 0969 | 0970 | 0971 | 0972 | 0973 | 0974 | 0975 |
| 3D0 | 0976 | 0977 | 0978 | 0979 | 0980 | 0981 | 0982 | 0983 | 0984 | 0985 | 0986 | 0987 | 0988 | 0989 | 0990 | 0991 |
| 3E0 | 0992 | 0993 | 0994 | 0995 | 0996 | 0997 | 0998 | 0999 | 1000 | 1001 | 1002 | 1003 | 1004 | 1005 | 1006 | 1007 |
| 3F0 | 1008 | 1009 | 1010 | 1011 | 1012 | 1013 | 1014 | 1015 | 1016 | 1017 | 1018 | 1019 | 1020 | 1021 | 1022 | 1023 |

## HEXADECIMAL-DECIMAL INTEGER CONVERSION (Cont'd)

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 400 | 1024 | 1025 | 1026 | 1027 | 1028 | 1029 | 1030 | 1031 | 1032 | 1033 | 1034 | 1035 | 1036 | 1037 | 1038 | 1039 |
| 410 | 1040 | 1041 | 1042 | 1043 | 1044 | 1045 | 1046 | 1047 | 1048 | 1049 | 1050 | 1051 | 1052 | 1053 | 1054 | 1055 |
| 420 | 1056 | 1057 | 1058 | 1059 | 1060 | 1061 | 1062 | 1063 | 1064 | 1065 | 1066 | 1067 | 1068 | 1069 | 1070 | 1071 |
| 430 | 1072 | 1073 | 1074 | 1075 | 1076 | 1077 | 1078 | 1079 | 1080 | 1081 | 1082 | 1083 | 1084 | 1085 | 1086 | 1087 |
| 440 | 1088 | 1089 | 1090 | 1091 | 1092 | 1093 | 1094 | 1095 | 1096 | 1097 | 1098 | 1099 | 1100 | 1101 | 1102 | 1103 |
| 450 | 1104 | 1105 | 1106 | 1107 | 1108 | 1109 | 1110 | 1111 | 1112 | 1113 | 1114 | 1115 | 1116 | 1117 | 1118 | 1119 |
| 460 | 1120 | 1121 | 1122 | 1123 | 1124 | 1125 | 1126 | 1127 | 1128 | 1129 | 1130 | 1131 | 1132 | 1133 | 1134 | 1135 |
| 470 | 1136 | 1137 | 1138 | 1139 | 1140 | 1141 | 1142 | 1143 | 1144 | 1145 | 1146 | 1147 | 1148 | 1149 | 1150 | 1151 |
| 480 | 1152 | 1153 | 1154 | 1155 | 1156 | 1157 | 1158 | 1159 | 1160 | 1161 | 1162 | 1163 | 1164 | 1165 | 1166 | 1167 |
| 490 | 1168 | 1169 | 1170 | 1171 | 1172 | 1173 | 1174 | 1175 | 1176 | 1177 | 1178 | 1179 | 1180 | 1181 | 1182 | 1183 |
| 4A0 | 1184 | 1185 | 1186 | 1187 | 1188 | 1189 | 1190 | 1191 | 1192 | 1193 | 1194 | 1195 | 1196 | 1197 | 1198 | 1199 |
| 4B0 | 1200 | 1201 | 1202 | 1203 | 1204 | 1205 | 1206 | 1207 | 1208 | 1209 | 1210 | 1211 | 1212 | 1213 | 1214 | 1215 |
| 4C0 | 1216 | 1217 | 1218 | 1219 | 1220 | 1221 | 1222 | 1223 | 1224 | 1225 | 1226 | 1227 | 1228 | 1229 | 1230 | 1231 |
| 4D0 | 1232 | 1233 | 1234 | 1235 | 1236 | 1237 | 1238 | 1239 | 1240 | 1241 | 1242 | 1243 | 1244 | 1245 | 1246 | 1247 |
| 4E0 | 1248 | 1249 | 1250 | 1251 | 1252 | 1253 | 1254 | 1255 | 1256 | 1257 | 1258 | 1259 | 1260 | 1261 | 1262 | 1263 |
| 4F0 | 1264 | 1265 | 1266 | 1267 | 1268 | 1269 | 1270 | 1271 | 1272 | 1273 | 1274 | 1275 | 1276 | 1277 | 1278 | 1279 |
| 500 | 1280 | 1281 | 1282 | 1283 | 1284 | 1285 | 1286 | 1287 | 1288 | 1289 | 1290 | 1291 | 1292 | 1293 | 1294 | 1295 |
| 510 | 1296 | 1297 | 1298 | 1299 | 1300 | 1301 | 1302 | 1303 | 1304 | 1305 | 1306 | 1307 | 1308 | 1309 | 1310 | 1311 |
| 520 | 1312 | 1313 | 1314 | 1315 | 1316 | 1317 | 1318 | 1319 | 1320 | 1321 | 1322 | 1323 | 1324 | 1325 | 1326 | 1327 |
| 530 | 1328 | 1329 | 1330 | 1331 | 1332 | 1333 | 1334 | 1335 | 1336 | 1337 | 1338 | 1339 | 1340 | 1341 | 1342 | 1343 |
| 540 | 1344 | 1345 | 1346 | 1347 | 1348 | 1349 | 1350 | 1351 | 1352 | 1353 | 1354 | 1355 | 1356 | 1357 | 1358 | 1359 |
| 550 | 1360 | 1361 | 1362 | 1363 | 1364 | 1365 | 1366 | 1367 | 1368 | 1369 | 1370 | 1371 | 1372 | 1373 | 1374 | 1375 |
| 560 | 1376 | 1377 | 1378 | 1379 | 1380 | 1381 | 1382 | 1383 | 1384 | 1385 | 1386 | 1387 | 1388 | 1389 | 1390 | 1391 |
| 570 | 1392 | 1393 | 1394 | 1395 | 1396 | 1397 | 1398 | 1399 | 1400 | 1401 | 1402 | 1403 | 1404 | 1405 | 1406 | 1407 |
| 580 | 1408 | 1409 | 1410 | 1411 | 1412 | 1413 | 1414 | 1415 | 1416 | 1417 | 1418 | 1419 | 1420 | 1421 | 1422 | 1423 |
| 590 | 1424 | 1425 | 1426 | 1427 | 1428 | 1429 | 1430 | 1431 | 1432 | 1433 | 1434 | 1435 | 1436 | 1437 | 1438 | 1439 |
| 5A0 | 1440 | 1441 | 1442 | 1443 | 1444 | 1445 | 1446 | 1447 | 1448 | 1449 | 1450 | 1451 | 1452 | 1453 | 1454 | 1455 |
| 5B0 | 1456 | 1457 | 1458 | 1459 | 1460 | 1461 | 1462 | 1463 | 1464 | 1465 | 1466 | 1467 | 1468 | 1469 | 1470 | 1471 |
| 5C0 | 1472 | 1473 | 1474 | 1475 | 1476 | 1477 | 1478 | 1479 | 1480 | 1481 | 1482 | 1483 | 1484 | 1485 | 1486 | 1487 |
| 5D0 | 1488 | 1489 | 1490 | 1491 | 1492 | 1493 | 1494 | 1495 | 1496 | 1497 | 1498 | 1499 | 1500 | 1501 | 1502 | 1503 |
| 5E0 | 1504 | 1505 | 1506 | 1507 | 1508 | 1509 | 1510 | 1511 | 1512 | 1513 | 1514 | 1515 | 1516 | 1517 | 1518 | 1519 |
| 5F0 | 1520 | 1521 | 1522 | 1523 | 1524 | 1525 | 1526 | 1527 | 1528 | 1529 | 1530 | 1531 | 1532 | 1533 | 1534 | 1535 |
| 600 | 1536 | 1537 | 1538 | 1539 | 1540 | 1541 | 1542 | 1543 | 1544 | 1545 | 1546 | 1547 | 1548 | 1549 | 1550 | 1551 |
| 610 | 1552 | 1553 | 1554 | 1555 | 1556 | 1557 | 1558 | 1559 | 1560 | 1561 | 1562 | 1563 | 1564 | 1565 | 1566 | 1567 |
| 620 | 1568 | 1569 | 1570 | 1571 | 1572 | 1573 | 1574 | 1575 | 1576 | 1577 | 1578 | 1579 | 1580 | 1581 | 1582 | 1583 |
| 630 | 1584 | 1585 | 1586 | 1587 | 1588 | 1589 | 1590 | 1591 | 1592 | 1593 | 1594 | 1595 | 1596 | 1597 | 1598 | 1599 |
| 640 | 1600 | 1601 | 1602 | 1603 | 1604 | 1605 | 1606 | 1607 | 1608 | 1609 | 1610 | 1611 | 1612 | 1613 | 1614 | 1615 |
| 650 | 1616 | 1617 | 1618 | 1619 | 1620 | 1621 | 1622 | 1623 | 1624 | 1625 | 1626 | 1627 | 1628 | 1629 | 1630 | 1631 |
| 660 | 1632 | 1633 | 1634 | 1635 | 1636 | 1637 | 1638 | 1639 | 1640 | 1641 | 1642 | 1643 | 1644 | 1645 | 1646 | 1647 |
| 670 | 1648 | 1649 | 1650 | 1651 | 1652 | 1653 | 1654 | 1655 | 1656 | 1657 | 1658 | 1659 | 1660 | 1661 | 1662 | 1663 |
| 680 | 1664 | 1665 | 1666 | 1667 | 1668 | 1669 | 1670 | 1671 | 1672 | 1673 | 1674 | 1675 | 1676 | 1677 | 1678 | 1679 |
| 690 | 1680 | 1681 | 1682 | 1683 | 1684 | 1685 | 1686 | 1687 | 1688 | 1689 | 1690 | 1691 | 1692 | 1693 | 1694 | 1695 |
| 6A0 | 1696 | 1697 | 1698 | 1699 | 1700 | 1701 | 1702 | 1703 | 1704 | 1705 | 1706 | 1707 | 1708 | 1709 | 1710 | 1711 |
| 6B0 | 1712 | 1713 | 1714 | 1715 | 1716 | 1717 | 1718 | 1719 | 1720 | 1721 | 1722 | 1723 | 1724 | 1725 | 1726 | 1727 |
| 6C0 | 1728 | 1729 | 1730 | 1731 | 1732 | 1733 | 1734 | 1735 | 1736 | 1737 | 1738 | 1739 | 1740 | 1741 | 1742 | 1743 |
| 6D0 | 1744 | 1745 | 1746 | 1747 | 1748 | 1749 | 1750 | 1751 | 1752 | 1753 | 1754 | 1755 | 1756 | 1757 | 1758 | 1759 |
| 6E0 | 1760 | 1761 | 1762 | 1763 | 1764 | 1765 | 1766 | 1767 | 1768 | 1769 | 1770 | 1771 | 1772 | 1773 | 1774 | 1775 |
| 6F0 | 1776 | 1777 | 1778 | 1779 | 1780 | 1781 | 1782 | 1783 | 1784 | 1785 | 1786 | 1787 | 1788 | 1789 | 1790 | 1791 |

## HEXADECIMAL-DECIMAL INTEGER CONVERSION (Cont'd)

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 700 | 1792 | 1793 | 1794 | 1795 | 1796 | 1797 | 1798 | 1799 | 1800 | 1801 | 1802 | 1803 | 1804 | 1805 | 1806 | 1807 |
| 710 | 1808 | 1809 | 1810 | 1811 | 1812 | 1813 | 1814 | 1815 | 1816 | 1817 | 1818 | 1819 | 1820 | 1821 | 1822 | 1823 |
| 720 | 1824 | 1825 | 1826 | 1827 | 1828 | 1829 | 1830 | 1831 | 1832 | 1833 | 1834 | 1835 | 1836 | 1837 | 1838 | 1839 |
| 730 | 1840 | 1841 | 1842 | 1843 | 1844 | 1845 | 1846 | 1847 | 1848 | 1849 | 1850 | 1851 | 1852 | 1853 | 1854 | 1855 |
| 740 | 1856 | 1857 | 1858 | 1859 | 1860 | 1861 | 1862 | 1863 | 1864 | 1865 | 1866 | 1867 | 1868 | 1869 | 1870 | 1871 |
| 750 | 1872 | 1873 | 1874 | 1875 | 1876 | 1877 | 1878 | 1879 | 1880 | 1881 | 1882 | 1883 | 1884 | 1885 | 1886 | 1887 |
| 760 | 1888 | 1889 | 1890 | 1891 | 1892 | 1893 | 1894 | 1895 | 1896 | 1897 | 1898 | 1899 | 1900 | 1901 | 1902 | 1903 |
| 770 | 1904 | 1905 | 1906 | 1907 | 1908 | 1909 | 1910 | 1911 | 1912 | 1913 | 1914 | 1915 | 1916 | 1917 | 1918 | 1919 |
| 780 | 1920 | 1921 | 1922 | 1923 | 1924 | 1925 | 1926 | 1927 | 1928 | 1929 | 1930 | 1931 | 1932 | 1933 | 1934 | 1935 |
| 790 | 1936 | 1937 | 1938 | 1939 | 1940 | 1941 | 1942 | 1943 | 1944 | 1945 | 1946 | 1947 | 1948 | 1949 | 1950 | 1951 |
| 7A0 | 1952 | 1953 | 1954 | 1955 | 1956 | 1957 | 1958 | 1959 | 1960 | 1961 | 1962 | 1963 | 1964 | 1965 | 1966 | 1967 |
| 7B0 | 1968 | 1969 | 1970 | 1971 | 1972 | 1973 | 1974 | 1975 | 1976 | 1977 | 1978 | 1979 | 1980 | 1981 | 1982 | 1983 |
| 7C0 | 1984 | 1985 | 1986 | 1987 | 1988 | 1989 | 1990 | 1991 | 1992 | 1993 | 1994 | 1995 | 1996 | 1997 | 1998 | 1999 |
| 7D0 | 2000 | 2001 | 2002 | 2003 | 2004 | 2005 | 2006 | 2007 | 2008 | 2009 | 2010 | 2011 | 2012 | 2013 | 2014 | 2015 |
| 7E0 | 2016 | 2017 | 2018 | 2019 | 2020 | 2021 | 2022 | 2023 | 2024 | 2025 | 2026 | 2027 | 2028 | 2029 | 2030 | 2031 |
| 7F0 | 2032 | 2033 | 2034 | 2035 | 2036 | 2037 | 2038 | 2039 | 2040 | 2041 | 2042 | 2043 | 2044 | 2045 | 2046 | 2047 |
| 800 | 2048 | 2049 | 2050 | 2051 | 2052 | 2053 | 2054 | 2055 | 2056 | 2057 | 2058 | 2059 | 2060 | 2061 | 2062 | 2063 |
| 810 | 2064 | 2065 | 2066 | 2067 | 2068 | 2069 | 2070 | 2071 | 2072 | 2073 | 2074 | 2075 | 2076 | 2077 | 2078 | 2079 |
| 820 | 2080 | 2081 | 2082 | 2083 | 2084 | 2085 | 2086 | 2087 | 2088 | 2089 | 2090 | 2091 | 2092 | 2093 | 2094 | 2095 |
| 830 | 2096 | 2097 | 2098 | 2099 | 2100 | 2101 | 2102 | 2103 | 2104 | 2105 | 2106 | 2107 | 2108 | 2109 | 2110 | 2111 |
| 840 | 2112 | 2113 | 2114 | 2115 | 2116 | 2117 | 2118 | 2119 | 2120 | 2121 | 2122 | 2123 | 2124 | 2125 | 2126 | 2127 |
| 850 | 2128 | 2129 | 2130 | 2131 | 2132 | 2133 | 2134 | 2135 | 2136 | 2137 | 2138 | 2139 | 2140 | 2141 | 2142 | 2143 |
| 960 | 2144 | 2145 | 2146 | 2147 | 2148 | 2149 | 2150 | 2151 | 2152 | 2153 | 2154 | 2155 | 2156 | 2157 | 2158 | 2159 |
| 870 | 2160 | 2161 | 2162 | 2163 | 2164 | 2165 | 2166 | 2167 | 2168 | 2169 | 2170 | 2171 | 2172 | 2173 | 2174 | 2175 |
| 880 | 2176 | 2177 | 2178 | 2179 | 2180 | 2181 | 2182 | 2183 | 2184 | 2185 | 2186 | 2187 | 2188 | 2189 | 2190 | 2191 |
| 890 | 2192 | 2193 | 2194 | 2195 | 2196 | 2197 | 2198 | 2199 | 2200 | 2201 | 2202 | 2203 | 2204 | 2205 | 2206 | 2207 |
| 8A0 | 2208 | 2209 | 2210 | 2211 | 2212 | 2213 | 2214 | 2215 | 2216 | 2217 | 2218 | 2219 | 2220 | 2221 | 2222 | 2223 |
| 8B0 | 2224 | 2225 | 2226 | 2227 | 2228 | 2229 | 2230 | 2231 | 2232 | 2233 | 2234 | 2235 | 2236 | 2237 | 2238 | 2239 |
| 8C0 | 2240 | 2241 | 2242 | 2243 | 2244 | 2245 | 2246 | 2247 | 2248 | 2249 | 2250 | 2251 | 2252 | 2253 | 2254 | 2255 |
| 8D0 | 2256 | 2257 | 2258 | 2259 | 2260 | 2261 | 2262 | 2263 | 2264 | 2265 | 2266 | 2267 | 2268 | 2269 | 2270 | 2271 |
| 8E0 | 2272 | 2273 | 2274 | 2275 | 2276 | 2277 | 2278 | 2279 | 2280 | 2281 | 2282 | 2283 | 2284 | 2285 | 2286 | 2287 |
| 8F0 | 2288 | 2289 | 2290 | 2291 | 2292 | 2293 | 2294 | 2295 | 2296 | 2297 | 2298 | 2299 | 2300 | 2301 | 2302 | 2303 |
| 900 | 2304 | 2305 | 2306 | 2307 | 2308 | 2309 | 2310 | 2311 | 2312 | 2313 | 2314 | 2315 | 2316 | 2317 | 2318 | 2319 |
| 910 | 2320 | 2321 | 2322 | 2323 | 2324 | 2325 | 2326 | 2327 | 2328 | 2329 | 2330 | 2331 | 2332 | 2333 | 2334 | 2335 |
| 920 | 2336 | 2337 | 2338 | 2339 | 2340 | 2341 | 2342 | 2343 | 2344 | 2345 | 2346 | 2347 | 2348 | 2349 | 2350 | 2351 |
| 930 | 2352 | 2353 | 2354 | 2355 | 2356 | 2357 | 2358 | 2359 | 2360 | 2361 | 2362 | 2363 | 2364 | 2365 | 2366 | 2367 |
| 940 | 2368 | 2369 | 2370 | 2371 | 2372 | 2373 | 2374 | 2375 | 2376 | 2377 | 2378 | 2379 | 2380 | 2381 | 2382 | 2383 |
| 950 | 2384 | 2385 | 2386 | 2387 | 2388 | 2389 | 2390 | 2391 | 2392 | 2393 | 2394 | 2395 | 2396 | 2397 | 2398 | 2399 |
| 960 | 2400 | 2401 | 2402 | 2403 | 2404 | 2405 | 2406 | 2407 | 2408 | 2409 | 2410 | 2411 | 2412 | 2413 | 2414 | 2415 |
| 970 | 2416 | 2417 | 2418 | 2419 | 2420 | 2421 | 2422 | 2423 | 2424 | 2425 | 2426 | 2427 | 2428 | 2429 | 2430 | 2431 |
| 980 | 2432 | 2433 | 2434 | 2435 | 2436 | 2437 | 2438 | 2439 | 2440 | 2441 | 2442 | 2443 | 2444 | 2445 | 2446 | 2447 |
| 990 | 2448 | 2449 | 2450 | 2451 | 2452 | 2453 | 2454 | 2455 | 2456 | 2457 | 2458 | 2459 | 2460 | 2461 | 2462 | 2463 |
| 9A0 | 2464 | 2465 | 2466 | 2467 | 2468 | 2469 | 2470 | 2471 | 2472 | 2473 | 2474 | 2475 | 2476 | 2477 | 2478 | 2479 |
| 9B0 | 2480 | 2481 | 2482 | 2483 | 2484 | 2485 | 2486 | 2487 | 2488 | 2489 | 2490 | 2491 | 2492 | 2493 | 2494 | 2495 |
| C0 | 2496 | 2497 | 2498 | 2499 | 2500 | 2501 | 2502 | 2503 | 2504 | 2505 | 2506 | 2507 | 2508 | 2509 | 2510 | 2511 |
| 9D0 | 2512 | 2513 | 2514 | 2515 | 2516 | 2517 | 2518 | 2519 | 2520 | 2521 | 2522 | 2523 | 2524 | 2525 | 2526 | 2527 |
| 9E0 | 2528 | 2529 | 2530 | 2531 | 2532 | 2533 | 2534 | 2535 | 2536 | 2537 | 2538 | 2539 | 2540 | 2541 | 2542 | 2543 |
| 9F0 | 2544 | 2545 | 2546 | 2547 | 2548 | 2549 | 2550 | 2551 | 2552 | 2553 | 2554 | 2555 | 2556 | 2557 | 2558 | 2559 |

## HEXADECIMAL-DECIMAL INTEGER CONVERSION (Cont'd)

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A00 | 2560 | 2561 | 2562 | 2563 | 2564 | 2565 | 2566 | 2567 | 2568 | 2569 | 2570 | 2571 | 2572 | 2573 | 2574 | 2575 |
| A10 | 2576 | 2577 | 2578 | 2579 | 2580 | 2581 | 2582 | 2583 | 2584 | 2585 | 2586 | 2587 | 2588 | 2589 | 2590 | 2591 |
| A20 | 2592 | 2593 | 2594 | 2595 | 2596 | 2597 | 2598 | 2599 | 2600 | 2601 | 2602 | 2603 | 2604 | 2605 | 2606 | 2607 |
| A30 | 2608 | 2609 | 2610 | 2611 | 2612 | 2613 | 2614 | 2615 | 2616 | 2617 | 2618 | 2619 | 2620 | 2621 | 2622 | 2623 |
| A40 | 2624 | 2625 | 2626 | 2627 | 2628 | 2629 | 2630 | 2631 | 2632 | 2633 | 2634 | 2635 | 2636 | 2637 | 2638 | 2639 |
| A50 | 2640 | 2641 | 2642 | 2643 | 2644 | 2645 | 2646 | 2647 | 2648 | 2649 | 2650 | 2651 | 2652 | 2653 | 2654 | 2655 |
| A60 | 2656 | 2657 | 2658 | 2659 | 2660 | 2661 | 2662 | 2663 | 2664 | 2665 | 2666 | 2667 | 2668 | 2669 | 2670 | 2671 |
| A70 | 2672 | 2673 | 2674 | 2675 | 2676 | 2677 | 2678 | 2679 | 2680 | 2681 | 2682 | 2683 | 2684 | 2685 | 2686 | 2687 |
| A80 | 2688 | 2689 | 2690 | 2691 | 2692 | 2693 | 2694 | 2695 | 2696 | 2697 | 2698 | 2699 | 2700 | 2701 | 2702 | 2703 |
| A90 | 2704 | 2705 | 2706 | 2707 | 2708 | 2709 | 2710 | 2711 | 2712 | 2713 | 2714 | 2715 | 2716 | 2717 | 2718 | 2719 |
| AA0 | 2720 | 2721 | 2722 | 2723 | 2724 | 2725 | 2726 | 2727 | 2728 | 2729 | 2730 | 2731 | 2732 | 2733 | 2734 | 2735 |
| AB0 | 2736 | 2737 | 2738 | 2739 | 2740 | 2741 | 2742 | 2743 | 2744 | 2745 | 2746 | 2747 | 2748 | 2749 | 2750 | 2751 |
| AC0 | 2752 | 2753 | 2754 | 2755 | 2756 | 2757 | 2758 | 2759 | 2760 | 4761 | 2762 | 2763 | 2764 | 2765 | 2766 | 2767 |
| AD0 | 2768 | 2769 | 2770 | 2771 | 2772 | 2773 | 2774 | 2775 | 2776 | 2777 | 2778 | 2779 | 2780 | 2781 | 2782 | 2783 |
| AE0 | 2784 | 2785 | 2786 | 2787 | 2788 | 2789 | 2790 | 2791 | 2792 | 2793 | 2794 | 2795 | 2796 | 2797 | 2798 | 2799 |
| AF0 | 2800 | 2801 | 2802 | 2803 | 2804 | 2805 | 2806 | 2807 | 2808 | 2809 | 2810 | 2811 | 2812 | 2813 | 2814 | 2815 |
| B00 | 2816 | 2817 | 2818 | 2819 | 2820 | 2821 | 2822 | 2823 | 2824 | 2825 | 2826 | 2827 | 2828 | 2829 | 2830 | 2831 |
| B10 | 2832 | 2833 | 2834 | 2835 | 2836 | 2837 | 2838 | 2839 | 2840 | 2841 | 2842 | 2843 | 2844 | 2845 | 2846 | 2847 |
| B20 | 2848 | 2849 | 2850 | 3851 | 2852 | 2853 | 2854 | 2855 | 2856 | 2857 | 2858 | 2859 | 2860 | 2861 | 2862 | 2863 |
| B30 | 2864 | 2865 | 2866 | 2867 | 2868 | 2869 | 2870 | 2871 | 2872 | 2873 | 2874 | 2875 | 2876 | 2877 | 2878 | 2879 |
| B40 | 2880 | 2881 | 2882 | 2883 | 2884 | 2885 | 2866 | 2887 | 2888 | 2889 | 2890 | 2891 | 2892 | 2893 | 2894 | 2895 |
| B50 | 2896 | 2897 | 2898 | 2899 | 2900 | 2901 | 2902 | 2903 | 2904 | 2905 | 2906 | 2907 | 2908 | 2909 | 2910 | 2911 |
| B60 | 2912 | 2913 | 2914 | 2915 | 2916 | 2917 | 2918 | 2919 | 2920 | 2921 | 2922 | 2923 | 2924 | 2925 | 2926 | 2927 |
| B70 | 2928 | 2929 | 2930 | 2931 | 2932 | 2933 | 2934 | 2935 | 2936 | 2937 | 2938 | 2939 | 2940 | 2941 | 2942 | 2943 |
| B80 | 2944 | 2945 | 2946 | 2947 | 2948 | 2949 | 2950 | 2951 | 2952 | 2953 | 2954 | 2955 | 2956 | 2957 | 2958 | 2959 |
| B90 | 2960 | 2961 | 2962 | 2963 | 2964 | 2965 | 2966 | 2967 | 2968 | 2969 | 2970 | 2971 | 2972 | 2973 | 2974 | 2975 |
| BA0 | 2976 | 2977 | 2978 | 2979 | 2980 | 2981 | 2982 | 2983 | 2984 | 2985 | 2986 | 2987 | 2988 | 2989 | 2990 | 2991 |
| B80 | 2992 | 2993 | 2994 | 2995 | 2996 | 2997 | 2998 | 2999 | 3000 | 3001 | 3002 | 3003 | 3004 | 3005 | 3006 | 3007 |
| BC0 | 3008 | 3009 | 3010 | 3011 | 3012 | 3013 | 3014 | 3015 | 3016 | 3017 | 3018 | 3019 | 3020 | 3021 | 3022 | 3023 |
| BD0 | 3024 | 3025 | 3026 | 3027 | 3028 | 3029 | 3030 | 3031 | 3032 | 3033 | 3034 | 3035 | 3036 | 3037 | 3038 | 3039 |
| BE0 | 3040 | 3041 | 3042 | 3043 | 3044 | 3045 | 3046 | 3047 | 3048 | 3049 | 3050 | 3051 | 3052 | 3053 | 3054 | 3055 |
| BF0 | 3056 | 3057 | 3058 | 3059 | 3060 | 3061 | 3062 | 3063 | 3064 | 3065 | 3066 | 3067 | 3068 | 3069 | 3070 | 3071 |
| C00 | 3072 | 3073 | 3074 | 3075 | 3076 | 3077 | 3078 | 3079 | 3080 | 3081 | 3082 | 3083 | 3084 | 3085 | 3086 | 3087 |
| C10 | 3088 | 3089 | 3090 | 3091 | 3092 | 3093 | 3094 | 3095 | 3096 | 3097 | 3098 | 3099 | 3100 | 3101 | 3102 | 3103 |
| C20 | 3104 | 3105 | 3106 | 3107 | 3108 | 3109 | 3110 | 3111 | 3112 | 3113 | 3114 | 3115 | 3116 | 3117 | 3118 | 3119 |
| C30 | 3120 | 3121 | 3122 | 3123 | 3124 | 3125 | 3126 | 3127 | 3128 | 3129 | 3130 | 3131 | 3132 | 3133 | 3134 | 3135 |
| C40 | 3136 | 3137 | 3138 | 3139 | 3140 | 3141 | 3142 | 3143 | 3144 | 3145 | 3146 | 3147 | 3148 | 3149 | 3150 | 3151 |
| C50 | 3152 | 3153 | 3154 | 3155 | 3156 | 3157 | 3158 | 3159 | 3160 | 3161 | 3162 | 3163 | 3164 | 3165 | 3166 | 3167 |
| C60 | 3168 | 3169 | 3170 | 3171 | 3172 | 3173 | 3174 | 3175 | 3176 | 3177 | 3178 | 3179 | 3180 | 3181 | 3182 | 3183 |
| C70 | 3184 | 3185 | 3186 | 3187 | 3188 | 3189 | 3190 | 3191 | 3192 | 3193 | 3194 | 3195 | 3196 | 3197 | 3198 | 3199 |
| C80 | 3200 | 3201 | 3202 | 3203 | 3204 | 3205 | 3206 | 3207 | 3208 | 3209 | 3210 | 3211 | 3212 | 3213 | 3214 | 3215 |
| C90 | 3216 | 3217 | 3218 | 3219 | 3220 | 3221 | 3222 | 3223 | 3224 | 3225 | 3226 | 3227 | 3228 | 3229 | 3230 | 3231 |
| CA0 | 3232 | 3233 | 3234 | 3235 | 3236 | 3237 | 3238 | 3239 | 3240 | 3241 | 3242 | 3243 | 3244 | 3245 | 3246 | 3247 |
| CB0 | 3248 | 3249 | 3250 | 3251 | 3252 | 3253 | 3254 | 3255 | 3256 | 3257 | 3258 | 3259 | 3260 | 3261 | 3262 | 3263 |
| CC0 | 3264 | 3265 | 3266 | 3267 | 3268 | 3269 | 3270 | 3271 | 3272 | 3273 | 3274 | 3275 | 3276 | 3277 | 3278 | 3279 |
| CD0 | 3280 | 3281 | 3282 | 3283 | 3284 | 3285 | 3286 | 3287 | 3288 | 3289 | 3290 | 3291 | 3292 | 3293 | 3294 | 3295 |
| CE0 | 3296 | 3297 | 3298 | 3299 | 3300 | 3301 | 3302 | 3303 | 3304 | 3305 | 3306 | 3307 | 3308 | 3309 | 3310 | 3311 |
| CF0 | 3312 | 3313 | 3314 | 3315 | 3316 | 3317 | 3318 | 3319 | 3320 | 3321 | 3322 | 3323 | 3324 | 3325 | 3326 | 3327 |

## HEXADECIMAL-DECIMAL INTEGER CONVERSION (Cont'd)

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| D00 | 3328 | 3329 | 3330 | 3331 | 3332 | 3333 | 3334 | 3335 | 3336 | 3337 | 3338 | 3339 | 3340 | 3341 | 3342 | 3343 |
| D10 | 3344 | 3345 | 3346 | 3347 | 3348 | 3349 | 3350 | 3351 | 3352 | 3353 | 3354 | 3355 | 3356 | 3357 | 3358 | 3359 |
| D20 | 3360 | 3361 | 3362 | 3363 | 3364 | 3365 | 3366 | 3367 | 3368 | 3369 | 3370 | 3371 | 3372 | 3373 | 3374 | 3375 |
| D30 | 3376 | 3377 | 3378 | 3379 | 3380 | 3381 | 3382 | 3383 | 3384 | 3385 | 3386 | 3387 | 3388 | 3389 | 3390 | 3391 |
| D40 | 3392 | 3393 | 3394 | 3395 | 3396 | 3397 | 3398 | 3399 | 3400 | 3401 | 3402 | 3403 | 3404 | 3405 | 3406 | 3407 |
| D50 | 3408 | 3409 | 3410 | 3411 | 3412 | 3413 | 3414 | 3415 | 3416 | 3417 | 3418 | 3419 | 3420 | 3421 | 3422 | 3423 |
| D60 | 3424 | 3425 | 3426 | 3427 | 3428 | 3429 | 3430 | 3431 | 3432 | 3433 | 3434 | 3435 | 3436 | 3437 | 3438 | 3439 |
| D70 | 3440 | 3441 | 3442 | 3443 | 3444 | 3445 | 3446 | 3447 | 3448 | 3449 | 3450 | 3451 | 3452 | 3453 | 3454 | 3455 |
| D80 | 3456 | 3457 | 3458 | 3459 | 3460 | 3461 | 3462 | 3463 | 3464 | 3465 | 3466 | 3467 | 3468 | 3469 | 3470 | 3471 |
| D90 | 3472 | 3473 | 3474 | 3475 | 3476 | 3477 | 3478 | 3479 | 3480 | 3481 | 3482 | 3483 | 3484 | 3485 | 3486 | 3487 |
| DA0 | 3488 | 3489 | 3490 | 3491 | 3492 | 3493 | 3494 | 3495 | 3496 | 3497 | 3498 | 3499 | 3500 | 3501 | 3502 | 3503 |
| DB0 | 3504 | 3505 | 3506 | 3507 | 3508 | 3509 | 3510 | 3511 | 3512 | 3513 | 3514 | 3515 | 3516 | 3517 | 3518 | 3519 |
| DC0 | 3520 | 3521 | 3522 | 3523 | 3524 | 3525 | 3526 | 3527 | 3528 | 3529 | 3530 | 3531 | 3532 | 3533 | 3534 | 3535 |
| CC0 | 3536 | 3537 | 3538 | 3539 | 3540 | 3541 | 3542 | 3543 | 3544 | 3545 | 3546 | 3547 | 3548 | 3549 | 3550 | 3551 |
| DE0 | 3552 | 3553 | 3554 | 3555 | 3556 | 3557 | 3558 | 3559 | 3560 | 3561 | 3562 | 3563 | 3564 | 3565 | 3566 | 3567 |
| DF0 | 3568 | 3569 | 3570 | 3571 | 3572 | 3573 | 3574 | 3575 | 3576 | 3577 | 3578 | 3579 | 3580 | 3581 | 3582 | 3583 |
| E00 | 3584 | 3585 | 3586 | 3587 | 3588 | 3589 | 3590 | 3591 | 3592 | 3593 | 3594 | 3595 | 3596 | 3597 | 3598 | 3599 |
| E10 | 3600 | 3601 | 3602 | 3603 | 3604 | 3605 | 3606 | 3607 | 3608 | 3609 | 3610 | 3611 | 3612 | 3613 | 3614 | 3615 |
| E20 | 3616 | 3617 | 3618 | 3619 | 3620 | 3621 | 3622 | 3623 | 3624 | 3625 | 3626 | 3627 | 3628 | 3629 | 3630 | 3631 |
| E30 | 3632 | 3633 | 3634 | 3635 | 3636 | 3637 | 3638 | 3639 | 3640 | 3641 | 3642 | 3643 | 3644 | 3645 | 3646 | 3647 |
| E40 | 3648 | 3649 | 3650 | 3651 | 3652 | 3653 | 3654 | 3655 | 3656 | 3657 | 3658 | 3659 | 3660 | 3661 | 3662 | 3663 |
| E50 | 3664 | 3665 | 3666 | 3667 | 3668 | 3669 | 3670 | 3671 | 3672 | 3673 | 3674 | 3675 | 3676 | 3677 | 3678 | 3679 |
| E60 | 3680 | 3681 | 3682 | 3683 | 3684 | 3685 | 3686 | 3687 | 3688 | 3689 | 3690 | 3691 | 3692 | 3693 | 3694 | 3695 |
| E70 | 3696 | 3697 | 3698 | 3699 | 3700 | 3701 | 3702 | 3703 | 3704 | 3705 | 3706 | 3707 | 3708 | 3709 | 3710 | 3711 |
| E80 | 3712 | 3713 | 3714 | 3715 | 3716 | 3717 | 3718 | 3719 | 3720 | 3721 | 3722 | 3723 | 3724 | 3725 | 3726 | 3727 |
| E90 | 3728 | 3729 | 3730 | 3731 | 3732 | 3733 | 3734 | 3735 | 3736 | 3737 | 3738 | 3739 | 3740 | 3741 | 3742 | 3743 |
| EA0 | 3744 | 3745 | 3746 | 3747 | 3748 | 3749 | 3750 | 3751 | 3752 | 3753 | 3754 | 3755 | 3756 | 3757 | 3758 | 3759 |
| EB0 | 3760 | 3761 | 3762 | 3763 | 3764 | 3765 | 3766 | 3767 | 3768 | 3769 | 3770 | 3771 | 3772 | 3773 | 3774 | 3775 |
| EC0 | 3776 | 3777 | 3778 | 3779 | 3780 | 3781 | 3782 | 3783 | 3784 | 3785 | 3786 | 3787 | 3788 | 3789 | 3790 | 3791 |
| ED0 | 3792 | 3793 | 3794 | 3795 | 3796 | 3797 | 3798 | 3799 | 3800 | 3801 | 3802 | 3803 | 3804 | 3805 | 3806 | 3807 |
| EE0 | 3808 | 3809 | 3810 | 3811 | 3812 | 3813 | 3814 | 3815 | 3816 | 3817 | 3818 | 3819 | 3820 | 3821 | 3822 | 3823 |
| EF0 | 3824 | 3825 | 3826 | 3827 | 3828 | 3829 | 3830 | 3831 | 3832 | 3833 | 3834 | 3835 | 3836 | 3837 | 3838 | 3839 |
| F00 | 3840 | 3841 | 3842 | 3843 | 3844 | 3845 | 3846 | 3847 | 3848 | 3849 | 3850 | 3851 | 3852 | 3853 | 3854 | 3855 |
| F10 | 3856 | 3857 | 3858 | 3859 | 3860 | 3861 | 3862 | 3863 | 3864 | 3865 | 3866 | 3867 | 3868 | 3869 | 3870 | 3871 |
| F20 | 3872 | 3873 | 3874 | 3875 | 3876 | 3877 | 3878 | 3879 | 3880 | 3881 | 3882 | 3883 | 3884 | 3885 | 3886 | 3887 |
| F30 | 3888 | 3889 | 3890 | 3891 | 3892 | 3893 | 3894 | 3895 | 3896 | 3897 | 3898 | 3899 | 3900 | 3901 | 3902 | 3903 |
| F40 | 3904 | 3905 | 3906 | 3907 | 3908 | 3909 | 3910 | 3911 | 3912 | 3913 | 3914 | 3915 | 3916 | 3917 | 3918 | 3919 |
| F50 | 3920 | 3921 | 3922 | 3923 | 3924 | 3925 | 3926 | 3927 | 3928 | 3929 | 3930 | 3931 | 3932 | 3933 | 3934 | 3935 |
| F60 | 3936 | 3937 | 3938 | 3939 | 3940 | 3941 | 3942 | 3943 | 3944 | 3945 | 3946 | 3947 | 3948 | 3949 | 3950 | 3951 |
| F70 | 3952 | 3953 | 3954 | 3955 | 3956 | 3957 | 3958 | 3959 | 3960 | 3961 | 3962 | 3963 | 3964 | 3965 | 3966 | 3967 |
| F80 | 3968 | 3969 | 3970 | 3971 | 3972 | 3973 | 3974 | 3975 | 3976 | 3977 | 3978 | 3979 | 3980 | 3981 | 3982 | 3983 |
| F90 | 3984 | 3985 | 3986 | 3987 | 3988 | 3989 | 3990 | 3991 | 3992 | 3993 | 3994 | 3995 | 3996 | 3997 | 3998 | 3999 |
| FA0 | 4000 | 4001 | 4002 | 4003 | 4004 | 4005 | 4006 | 4007 | 4008 | 4009 | 4010 | 4011 | 4012 | 4013 | 4014 | 4015 |
| FB0 | 4016 | 4017 | 4018 | 4019 | 4020 | 4021 | 4022 | 4023 | 4024 | 4025 | 4026 | 4027 | 4028 | 4029 | 4030 | 4031 |
| FC0 | 4032 | 4033 | 4034 | 4035 | 4036 | 4037 | 4038 | 4039 | 4040 | 4041 | 4042 | 4043 | 4044 | 4045 | 4046 | 4047 |
| FD0 | 4048 | 4049 | 4050 | 4051 | 4052 | 4053 | 4054 | 4055 | 4056 | 4057 | 4058 | 4059 | 4060 | 4061 | 4062 | 4063 |
| FE0 | 4064 | 4065 | 4066 | 4067 | 4068 | 4069 | 4070 | 4071 | 4072 | 4073 | 4074 | 4075 | 4076 | 4077 | 4078 | 4079 |
| FF0 | 4080 | 4081 | 4082 | 4083 | 4084 | 4085 | 4086 | 4087 | 4088 | 4089 | 4090 | 4091 | 4092 | 4093 | 4094 | 4095 |

# ASSEMBLER INSTALLATION NOTES

These notes are designed to help the user install the Assembler and make any modifications needed for a particular computer. The notes are separated into six sections: Program Installation; Program Modifications; Program I/O; Memory Requirements and Overlays; Cross Reference Notes; and NOVA Modifications.

## A. Program Installation

1. Do not hesitate to contact Microtec to discuss any Installation problems and any subsequent changes to the program. Because we have experience in the installation of our programs on most computers, we can advise the user of any potential problems.

2. The Assembler should be compiled once and its object module stored on some secondary device (disk). Compile the program, as any Fortran program, assigning it a name which can be refeenced by an Execute or Run command for the computer. If upon loading the compiled program it is discovered that there is insufficient main memory to hold the entire program, refer to the section describing Overlay Structures.

3. The user may compile the program as one entire program or as individual subroutines. It is easier to make changes to the program when each subroutine is compiled separately and linked together.

4. It is recommended that the user install and test the program as received from Microtec before making any changes to the program. It is desirable not to incorporate any program updates (if any) until the program is installed and tested as received.

## B. Program Modifications

1. Some computers do not accept the full ASCII character set. Therefore, some of the characters defined in Subroutine INIT may be illegal and give a compilation error. If this is the case on your computer, the illegal characters must be replaced by legal characters. If the characters are not used in the microprocessor assembly language (e.g. double quote), they may be replaced with blanks. If the illegal characters are used in the assembly language (e.g. greater than sign), replace each illegal character with a unique legal character and use the new character in place of the old character. The character arrays that need to be changed are in Subroutine INIT and are marked with comments.

2. The variable IBIT corresponds to the number of bits per word in the host computer. IBIT is initially set to 16. This variable determines how may characters are packed into one host computer word for symbols stored in the Assembler symbol table. The user may want to increase this variable if the host computer has a longer word length. Increasing IBIT will allow a larger number of symbols to be stored in

a given amount of memory. When initially installing the program, it is suggested that IBIT be left at 16 until the program is known to be operating correctly.

3. To increase the size of the symbol table and thus the number and length of the symbols that symbol table can hold, the user must change certain variables. The variables that must be changed depend on the nummber of in the host computer word (see 2.), the number of symbols in the symbol table, and the number of characters used to define a symbol. The variables that define these parameters are as follows:

```
IBIT  - number of bits per host computer word (set by user in INIT)
MLAB  - maximum label length in characters (set by user in INIT)
ICCNT - number of characters per host computer word (calculated)
IWORD - number of computer words per symbol (calculated)
LTAB  - length of symbol table (set by user in INIT)
```

The user must change the following variables to reflect the size of the symbol table and the length of a symbol. The arrays are in COMMON, and therefore, the dimensions need to be changed in every Subroutine.

```
ITAB(IWORD,LTAB)          where: IWORD = 1+(MLAB-1)/ICCNT
ITABV(LTAB)                      ICCNT = IBIT/8
ITABS(LTAB)
NAME(IWORD)
```

4. To increase the number of macros that may be defined, the following variables must be modified. These arrays are in Common, and therefore, the dimensions need to be changed in every Subroutine.

```
MXMAC - maximum macro count (set by user in INIT)
MDISK(MXMAC)
MPARC(MXMAC)
MCNAM(IWORD,MXMAC)
```

5. The number of columns of the input source statement that is written to the output listing is defined by the variable MLCOL in Subroutine INIT. MLCOL should be set to the maximum width of the users list output device minus 35 (width-35). The maximum value of MLCOL is 80 which corresponds to the full source statement. The default value of MLCOL is 72.

6. Users who need to use tabs in their source program should set the variable NCTAB in Subroutine INIT to a tab for their computer. It is now set to a blank. The program does not expand tabs on output, it only recognizes them as a valid delimiter. For most 16 bit minicomputers, the tab has a value of 9 and may be in the upper or lower 8 bits of the computer word with either blank or zero fill. Typical values for NCTAB would be: 9*256+32 (upper 8 bits blank fill) or 9 (lower 8 bits zero fill).

7. For various reasons some users may have to use labeled COMMON instead of normal blank COMMON (e.g. NOVA). When adding the COMMON block name to the COMMON statements, the statement must not exceed

column 72. If the user modifies any COMMON statements, the variables must be kept in the same order because of certain relationships within the program. Contact Microtec for any additional details.

C. Program Input/Output

1. The logical I/O device assignments assumed in the Assembler are:

```
KTERM = 1   (list device for error statements if used)
IPCH  = 4   (object module output device)
ICRD  = 5   (source statement input device)
IPRT  = 6   (output listing device)
IMFLE = 7   (intermediate file, disk or tape)
MCFLE = 8   (macro source file, disk)
```

These device assignments may have to be changed for your system. This may be done in either the Job Control Stream or in the program itself. If the device assignments are to be changed in the program, the variables may be found in Subroutine INIT.

2. Reading and writing to a bulk storage device such as a disk is not standard in FORTRAN. There are however, two usual methods to perform these operations. Method 1 uses a DEFINE FILE statement and standard READ and WRITE statements as follows:

```
DEFINE FILE 7(1000,97,U,IMREC)
WRITE(IMFLE'IMREC)  LIST
READ (IMFLE'IMREC)  LIST
```

where:
- IMFLE = 7 - is the file number of logical device
- 1000 - is the maximum number of records
- 97 - is the record size in words
- U - indicates a binary record
- IMREC - indicates the record number (associated variable)
- LIST - list of variables to read or write

Method 2 uses a CALL to an executive or system routine to process the disk read or write. For a typical computer this is as follows:

```
CALL EXEC(#,CODE,IBUF,CNT,NAME,IMREC)
```

where:
- # - indicates the type of call, read or write
- CODE - indicates binary or formatted I/O etc.
- IBUF - start of variables to read or write
- NAME - is typically a dimensioned array which contains the name of the disk file. This name is then used in the Job Control Stream to allocate disk storage.
- IMREC - disk record number

The Assembler Program uses Method 1 above as the standard method. However, statements for Method 2 are included in the program as comment statements for informative purposes.

3. All Program I/O activity except for generation of the output listing is done in Subroutine INOUT. This includes the reads and writes of the Intermediate files, reading the source input, and writing the output object module.

4. There are alternative ways of passing relocatable object modules from the Assembler to the Loader. The relocatable object modules could be written to a card punch, paper tape punch, or a tape unit by the Assembler, and read back by the Loader. Or, the relocatable modules could be saved on disk files by the Assembler. If object modules are passed from the Assembler to the Loader via disk files, the user must choose how to name the relocatable object module files generated by the Assembler. Three alternative methods are:

a. The assembler produces the object module on the same file during each assembly. The user must rename the file before another assembly is performed. Usually this can be easily done with a RENAME command in the assembler's Job Control Stream.

b. The Assembler writes the object module to a logical unit number, IPCH, but an ASSIGN or equivalent System Command is usedto equate the logical unit number with a disk file name. The user can vary the file name on the Command with each assembly run.

c. The Assembler can read the object module file name from an input device and open the specified disk file from the assembler program. If this is done, the file name must be read into an array with a pointer to the array in the system call that opens the file, or in the calls that read and write the file.

The assembler program currently writes the object module to a logical device. If the user wishes to open a disk file for the object module from the program, the user must add the necessary code.

5. As previously mentioned, the object module is written to logical device 4. The object record that is written to this unit is contained in array IPBUF which is padded out with blanks to 72 positions. The variable IPLEN indicates how many positions actually contain load information and should be used in a write statement to a sequential file or a paper tape unit to conserve space. This is the output statement used in Subroutine INOUT by the assembler. When writing to an I/O device that requires fixed length records (many disk units), use the complete 72 positions of array IPBUF. The DEFINE FILE statement shown as a comment in the Main Program for unit 4, and the disk write statements described in 2. above may have to be used. The object module disk file write may be formatted or unformatted (binary) as long as the read statement in the Loader performs a similar operation.

6. Some examples are shown below of system calls that open disk files and equate a logical device number 4 to the disk file name. If your computer uses these or similar statements they should be placed in the Main Program where the DEFINE FILE statement for logical device 4 is.

NOVA

```
CALL OPEN(4,"OBJECT",3,IER)
```

PDP-11 (RT-11)

```
CALL ASSIGN(4,"OBJECT")
```

On some computers, it is easier to assign room for, and name a disk file in the Job Control Stream preceding the assembly. No call OPEN is required for a file that already exists, and equating the file to a logical device is not necessary. The name of the file is placed in an array is Subroutine INOUT and the array name is placed in a system call.

HP 2100

```
:ST,B,OBJECT,50                              (in Control Stream)
:CALL EXEC(15,1091,IPBUF,72,NAMEP,IOREC)     (in INOUT, NAMEP contains
                                              name of file)
```

7. Some systems require disk space to be allocated for the temporary files used by the assembler, by placing statements in the Job Control Stream. Check to see if this is necessary for your system. The intermediate file (IMFLE) is used to store the source between passes of the Assembler. The macro file (MCFLE) is used to store the macro definitions. If macros are not used, the macro file need not be allocated (however, see section on cross reference table). It should be noted that the intermediate file could be replaced by any sequential file, such as a magnetic tape. However, the macro file requires a random access device. If IMFLE is a sequential file, then a REWIND IMFLE statement should be placed in the main program after the CALL PASS1 statement. The statement is already there noted by a comment.

8. Statements are included in subroutine INOUT for handling an end-of-file (EOF) condition for the input source. A typical READ statement with the END condition specified as shown in INOUT with a comment can be used on most machines. Some computers require a CALL to a system routine or some other statement to detect the EOF. Comments in Subroutine INOUT are included to show where the program should branch if an EOF is detected. Note: no modifications need be made to the INOUT Subroutine if EOF's are not detected.

9. The size of the Assembler's Intermediate file as well as program execution time may be reduced by scanning off source line trailing blanks before the source line is written to the Intermediate file and restoring them on the subsequent read. To do this, the intermediate file must be a sequential file and allow variable length records. The WRITE to the Intermedate file at statement 500 in Subroutine INOUT already scans off trailing blanks. However, a fixed length record is written to the file. To implement a variable length write the following statements must be changed near statement 500.

```
510   CONTINUE
520   KLEN = LTBLK+17
      WRITE(IMFLE) KLEN,(IMBUF(I),I=1,KLEN)
```

The following changes must be made for the Intermediate file read at statement number 200 in Subroutine INOUT.

```
200   READ(IMFLE) KLEN,(IMBUF(I),I=1,KLEN)
      IF(LTBLK-72) 210,230,230
210   LTBLK = LTBLK+1
      DO 220 I=LTBLK,72
      IN(I) = IBLNK
220   CONTINUE
230   RETURN
```

The READ and WRITE statements to the Intermediate file may have to be broken up into two pairs of statements to ensure that the variable KLEN is known before the list of variables is read. Whether or not any of the above changes are made, trailing blanks are not printed in the output listing routine.

10. When modifying any file definition statements such as the DEFINE FILE statements, or some equivalent statement, keep in mind that some statements require the record length in words and some in bytes. Typically large computers, e.g. IBM 370, CDC 6600, require these lengths in words, while most minicomputers require bytes. The user should check the correct value for his statement.

11. The user may list errors in the program to a separate I/O device (file) other than the standard listing device. The I/O device number is set by the variable KTERM in Subroutine INIT. If the user has specified "LIST E" in the assembly program, errors will be directed to this device. If this device is the same as the standard listing device, the error will not appear twice. If not used, it is recommended that this variable be set to the same as IPRT.

12. The size of the fixed length records written to the macro file (and cross reference) is set to 128 words. For users whose record size must be smaller than 128, the variable MSIZE in Subroutine INIT should be set to the smaller size. Users who can write larger length records should not increase the size.

D. Memory Requirements, Overlays, and Chaining

1. If core size is limited, the Assembler program may have to be Overlayed. One Overlay structure is as follows:

| Main | 1st Overlay | 2nd Overlay |
|---|---|---|
| MAIN | INIT | PASS2 |
| SCAN | PASS1 | LOUT |
| LABEL | OPCOD | OUT |
| SYMBL | MSCAN | ROUT |
| CONST | MCDEF | AOUT |
| XREFT | MCREF | SYMTA |
| INPUT | | CONVT |
| | | AVHEX |

A second structure which requires more overlays but reduces memory requirements even more is to place INIT in its own overlay. All routines shown in the 1st overlay above, except INIT, in another overlay. PASS2, LOUT, OUT, ROUT, and AOUT in another overlay, and SYMTA in a final overlay. In this case CONVT and AVHEX should be placed in the Main Segment.

2. If the user is not using the absolute object module facility via the "LIST A" directive, the AOUT Subroutine may be removed. In this case it is recommended that the Subroutine merely consist of a RETURN statement. If necesssary, the complete routine may be removed and the CALL to the AOUT routine from the start of the OUT routine should then be removed.

3. If a chaining facility is available, the routines shown above in the Main Segment may be compiled and loaded with the group of routines in the two overlays, creating two separate programs. The chain command may be used to call in the second program. If this is done, COMMON must usually be saved in the first program on a file and restored in the second program. On some computers this is automatically done by the chaining facility.

4. To aid those users who need to form their own Overlays or to segment their programs, the following list shows each routine in the assembler and all the routines that call it.

```
MAIN  -
INIT  - MAIN
INOUT - PASS1,PASS2,MCDEF,MCREF,ROUT,AOUT,SYMTA,XREFT
PASS1 - MAIN
PASS2 - MAIN
OPCOD - PASS1,MCDEF
LABEL - INIT,PASS1,PASS2
SYMBL - PASS1,OPCOD,LABEL,MCDEF,MSCAN
SCAN  - PASS1,PASS2
CONST - SCAN
MCDEF - PASS1
MCREF - PASS1
MSCAN - PASS1,MCDEF
LOUT  - PASS2
OUT   - PASS2
ROUT  - OUT
AOUT  - OUT
SYMTA - MAIN
XREFT - PASS2,LABEL,SYMTA
CONVT - ROUT,AOUT,SYMTA
AVHEX - LOUT,AOUT,SYMTA
```

E. Cross Reference

1. The cross reference table is accumulated in a memory array and when the array is filled it is stored on a disk file. However, the variable IXPAG in Subroutine INIT may be set to 0, in which case, if the memory array becomes full, the table will not be written to the disk file and further references will not be accumulated, but assembly will continue.

2. The standard cross reference table array size is 512 words. Each reference requires 2 words, hence 256 references may be accumulated in memory. To avoid using the disk, this array may be increased. The variables to change are described below.

3. The number of cross reference table arrays (memory array of 512 words) that will be written to the disk is set at 25. Hence 25*256 references can be accumulated with the standard program. This may also be increased.

4. The disk file that is used to store cross references (if necessary) is the same as the file used for macros. If necessary, for some reason, another file may be assigned. The cross reference READ and WRITE statements in INOUT may then have to be changed.

5. To increase the page size (memory array) of the cross reference table or total number of pages produced, or to not use the disk to store references, the following variables must be changed in Subroutine INIT.

MXREF - size of cross reference memory. The number of references on a page is (MXREF/2). MXREF should be divisible by 128.
IXTAB - cross reference array. Should be set to IXTAB(MXREF)
IXPAG - total number of pages of size MXREF that will be written before accumulation of references stops. If IXPAG=0 then no pages will be written to disk and references will only be accumulated in memory.

F. NOVA Modifications

When installing the Assembler on a NOVA computer, it is suggested that FORTRAN V be used. If FORTRAN IV is used, some additional program modifications have to be made.

1. Most versions of NOVA FORTRAN fill an H DATA specification statement with zeros and not blanks, as is typically done. Therefore characters read in under A formats must have the padded blanks stripped off. Insert the following statements after Fortran Statement 100 in Subroutine INOUT.

```
      DO 105 I=1,80
      IN(I) = IN(I).AND.-256
105   CONTINUE
```

2. All variables initialized in DATA statements must be placed in Labeled COMMON. The variables are local to each Subroutine, so unique dummy labels may be used for the COMMON Block names.

3. The DEFINE FILE statements in the Main Program must be replaced with CALL OPEN statements similar to those shown below.

```
      CALL OPEN(7,"IDUM1",3,IER)
      CALL OPEN(8,"IDUM2",3,IER,256)
```

The number of bytes per record must be included for random access files.

4. The Assembler Macro file must be random access, so a CALL to FSEEK must preceed each Macro and Cross Reference file access. Use Binary READ and WRITE statements for the intermediate file. To implement the above, change the FORTRAN source code in INOUT as follows:

```
200   READ BINARY(IMFLE) IMBUF

300   CALL FSEEK(MCFLE,MCREC)
      READ BINARY(MCFLE) MCBUF

400   CALL FSEEK(MCFLE,MCREC)
      READ BINARY(MCFLE) MCORE

500   WRITE BINARY(IMFLE) IMBUF

600   CALL FSEEK(MCFLE,MCREC)
      WRITE BINARY(MCFLE) MCBUF

700   CALL FSEEK(MCFLE,MCREC)
      WRITE BINARY(MCFLE) (IXTAB(J),J=1,128)
```

5. Several characters cannot be used in Hollerith DATA statements since they are not in the NOVA assembler's legal character set. These include the right and left parenthesis, percent sign, quote mark, etc.

Check your Assembly Language Manual for the legal character set. In Subroutine INIT replace all illegal characters with their internal numeric representation as they would appear if allowed in an 1H DATA format. This is shown below.

```
DATA NALPH( 1),NALPH( 2),NALPH( 3),NALPH( 4) /1H0,1H1,1H2,1H3/
                          .
                          .
                          .
DATA NALPH(33),NALPH(34),NALPH(35),NALPH(36) /1HW,1HX,1HY,1HZ/
DATA NALPH(37),NALPH(38),NALPH(39),NALPH(40) /16128,1H@,24320,1H /
DATA NALPH(41),NALPH(42),NALPH(43),NALPH(44) /1H!,8704,1H#,9216/
DATA NALPH(45),NALPH(46),NALPH(47),NALPH(48) /9472,1H&,9984,10240/
DATA NALPH(49),NALPH(50),NALPH(51),NALPH(52) /10496,1H*,1H+,1H,/
DATA NALPH(53),NALPH(54),NALPH(55),NALPH(56) /1H-,1H.,1H/,1H:/
DATA NALPH(57),NALPH(58),NALPH(59),NALPH(60) /1H;,15360,1H=,15872/

DATA NBLNK,NQUOT,NPLUS,NMIN,NGRAT,NLESS /1H ,9984,1H+,1H-,15872,15360/
DATA NDOLR,NCOMM,NAST,NSEMI,NCOLN       /9216,1H,,1H*,1H;,1H:/
DATA NDIV,NRPAR,NLPAR,NSHRP,NEQUL       /1H/,10496,10240,1H#,1H=/
DATA NCTAB,NPERC,NPER,NCAT,NAMP,NQUES   /2304,9472,1H.,1H@,1H&,16128/
```

6. On some NOVA systems any formatted records (list output and object output) written to a disk file, will lose the first character of the record (used as control character). If this is listed at a standard line printer it will act as the carriage control. But if punched on say paper tape, it will be lost. In this case the necesssary format statements in Subroutine INOUT should be modified to place a blank in the output record before each line.

# 6500
# LINKING LOADER MANUAL

Microtec
P.O. Box 60337
Sunnyvale, CA. 94088
408-733-2919

F. NOVA Modifications

When installing the Loader on a NOVA computer, it is suggested that FORTRAN V be used. If FORTRAN IV is used, some additional program modifications have to be made.

1. Most versions of NOVA FORTRAN fill an H DATA specification statement with zeros and not blanks, as is typically done. Therefore characters read under A formats must have the padded blanks stripped off. Insert the following statements after Fortran Statement 100 in Subroutine INOUT.

```
      DO 105 I=1,80
      IN(I) = IN(I).AND.-256
105   CONTINUE
```

2. All variables initialized in DATA statements must be placed in Labeled COMMON. The variables are local to each Subroutine, so unique dummy labels may be used for the COMMON Block names.

3. The DEFINE FILE statement in the Main Program must be replaced with a CALL OPEN statement similar to the one shown below.

```
      CALL OPEN(7,"IDUM1",3,IER)
```

4. Binary READ and WRITE statements should be used for the L to FSEEK intermediate file. To implement this change the FORTRAN source code in Subroutine INOUT should be as follows:

```
300   READ BINARY(IMFLE) IMBUF

400   WRITE BINARY(IMFLE) IMBUF
```

5. Several characters cannot be used in Hollerith DATA statements since they are not in the NOVA assembler's legal character set. Check your Assembly Language Manual for the legal character set. In Subroutine INIT replace all illegal characters with their internal numeric representation as they would appear if allowed in an 1H DATA format. This is shown below.

```
DATA NALPH( 1),NALPH( 2),NALPH( 3),NALPH( 4) /1H0,1H1,1H2,1H3/
                         .
                         .
                         .
DATA NALPH(33),NALPH(34),NALPH(35),NALPH(36) /1HW,1HX,1HY,1HZ/
DATA NALPH(37),NALPH(38),NALPH(39),NALPH(40) /16128,1H@,24320,1H /
DATA NCAT,NQUES,NAST,NBLNK,NCOMM          /1H@,16128,1H*,1H ,1H,/
DATA NCOLN,NEQUL,NDOLR,NSEMI              /1H:,1H=,9216,1H;/
```

6. On some NOVA systems any formatted records (list output and object output) written to a disk file, will lose the first character of the record (used as control character). If this is listed at a standard

line printer it will act as the carriage control. But if punched on paper tape, for example, it will be lost. In this case the necesssary format statements in Subroutine INOUT should be modified to place a blank in the output record before each line.

A simplified EQUAT Subroutine for PDP-11 computers is shown below. This Subroutine may be used to replace the EQUAT Subroutine currently in the Loader.

```
      LOGICAL*1  JNAME(18)

      REAL               leave this statements the same as in
      COMMON             current EQUAT subroutine


      IERR = 1
      K = 1
      DO 10 I=1,20
      IGBUF(I) = IBLNK
10    CONTINUE
100   IF((INC(JCOL).EQ.IBLNK .OR. (INC(JCOL).EQ.ICOMM)) GO TO 200
      IF(K .GT. 18) GO TO 900
      JNAME(K) = INC(JCOL)
      IGBUF(K) = INC(JCOL)
      K = K+1
      JCOL = JCOL+1
      GO TO 100
200   JNAME(K) = IBLNK
      CALL CLOSE(IFIL)
      CALL ASSIGN(IFIL,JNAME,0,'OLD')
      IRDR = IFIL
      IERR = 0
900   RETURN
      END
```

TABLE OF CONTENTS


| | | |
|---|---|---|
| 1.0 | INTRODUCTION | 1-1 |
| 2.0 | LOADER OPERATION | 2-1 |
| | Relocation Types | 2-3 |
| | Base Page | 2-4 |
| 3.0 | LOADER COMMANDS | 3-1 |
| | PSCT | 3-3 |
| | DSCT | 3-4 |
| | CSCT | 3-5 |
| | BSCT | 3-6 |
| | STACK | 3-7 |
| | MEMORY | 3-8 |
| | PPAGE | 3-9 |
| | DPAGE | 3-10 |
| | CPAGE | 3-11 |
| | ORDER | 3-12 |
| | START | 3-13 |
| | STKLN | 3-14 |
| | NAME | 3-15 |
| | LOAD | 3-16 |
| | XDEF | 3-17 |
| | LIST | 3-18 |
| | NLIST | 3-19 |
| | EXIT | 3-20 |
| | END | 3-21 |
| | Comment | 3-22 |
| 4.0 | HOW TO USE THE LOADER | 4-1 |
| | The Loader | 4-1 |
| | Loader Listing | 4-1 |
| | Loader Example | 4-4 |
| | APPENDIX A - Loader Messages | 5-1 |

# INTRODUCTION

This manual describes Microtec's 6500 Linking Loader that accompanies the 6500 Relocatable Assembler. The Linking Loader can be used to combine several independently assembled relocatable object modules into a single absolute object module. External references between modules are resolved and the final absolute symbol value substituted for each reference.

The Loader not only provides for the linking of several modules and adjusting the relocatable addresses into absolute addresses, but allows the program segment addresses to be specified, external definitions to be defined, the final load address and segment loading order to be specified the program segments.

# LOADER OPERATION

Many programs are too long to conveniently assemble as a single module. To avoid long assembly times, or to reduce the symbol table size, programs can be subdivided into smaller modules, assembled separately, and linked together by this loader program. After the separate program modules are linked and loaded by this program, the output module functions as if it had been generated by a single assembly.

The primary functions of the Linking Loader are:

1. To resolve external references between modules and check for undefined references (linking)
2. To adjust all relocatable addresses to the proper absolute addresses (loading)
3. To output the final absolute object module

To understand the loading process and to enable the user to use the Assembler and Linking Loader (hereafter called Loader) effectively, the user should understand the various program segments and segment load addresses. Although described in the Assembler Manual, the various segments are summarized below.

Absolute Segment - this segment is that part of the assembly program that contains no relocatable information but is to be loaded at fixed locations in the users memory. Absolute code is placed into the output module exactly as it is read from the input object modules.

Program Segment - the Program Segment contains that part of the program which comprises actual machine instructions and which typically can be placed into ROM. Instructions in the Program Segment can make reference to any other segments.

Data Segment - the Data Segment contains specifications for that part of a users program that typically contains run time data which usually is destined to reside in RAM. Nothing restricts this segment from containing executable instructions.

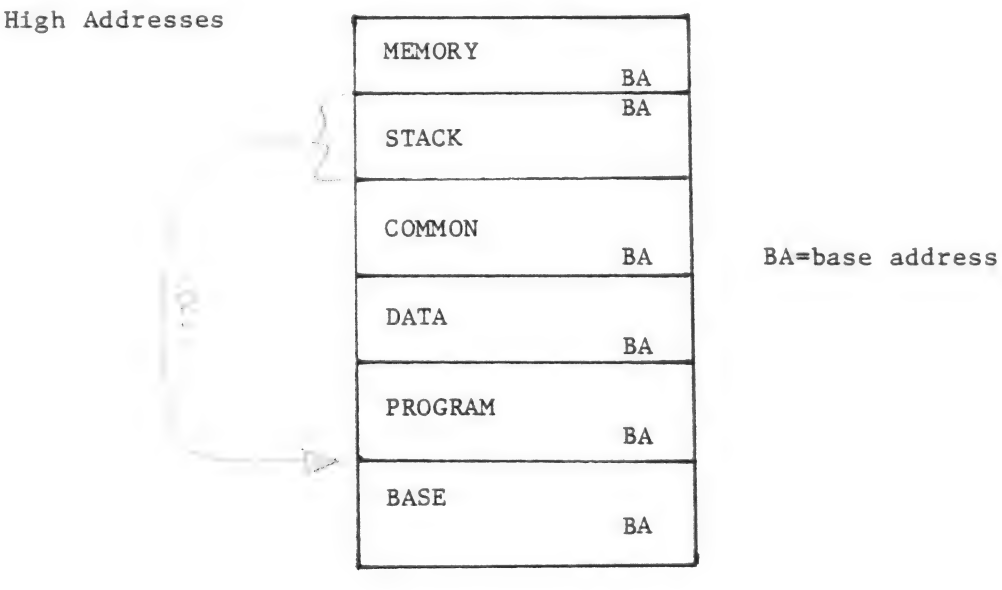
Common Segment - the Common Segment contains variables that may be referenced by each module. All Common Segments are located at the same address and thus provide for an effective communication area. This is similar to FORTRAN Common. Variables in the Common Segment may not be initialized.

Base Segment - The Base Segment which is assumed to reside within memory addresses 0 to OFFH may contain both data and instructions. It is a separate segment due to the fact that instructions referencing this segment require only 2 bytes instead of 3 bytes.

Stack Segment - the Stack Segment is used as the 6500 run time stack during program execution.

Memory Segment - the Memory Segment is usually the high address portion of memory which is not allocated to any of the other segments. Data tables may expand into the memory segment but the assembler has no facility to cause instructions to be loaded into the Memory Segment. The start of the Memory Segment is determined at Load time.

The Loader allows the user to load the segments into a contiguous program module or to specify the starting address (Base Address) of any or all of the segments. The order in which the segments are to be placed in memory may also be specified. The default organization used by the Loader is shown below.




Typically this is the memory organization used in most programs, but some users may want to place the COMMON and STACK segments after the PROGRAM segment so that the DATA segment can expand into the MEMORY segment during program execution.

The Base Address (BA) for all segments except the STACK segment is the low address of the segment. When a user specifies the starting address of a segment with a Loader command, it is the Base Address that is being specified. The Base Address for the STACK segment is the high address of the segment. This is done because the stack pointer typically moves toward lower addresses during program execution.

The user will typically have to set the starting address of the Stack Segment, if used, because the 6500 microprocessor requires the stack to reside between 100H and 1FFH in the memory space. The default segment organization may or may not cause this to happen depending upon the segment lengths. The Loader will issue a warning message indicating the Stack Segment does not reside on page 1 if this is the case. This message may be displayed even if the user does not make reference to the Stack Segment in the program.

## Relocation Types

The relocation type of any program segment is determined in the Assembler by the PSCT,DSCT, or CSCT directives. They may also be specified by Loader Commands. The effect of the three relocation types in the Loader are explained below.

Byte Relocation - this implies that no operand was specified on the segment directive in the assembler. In this case the segment from the object module will be placed immediately after the same segment from the preceding object module and there will be no mmemory wasted. For the Common Segment, it implies that the segment will be located immediately after the preceding Common segment, unless a starting address has been specified.

Page Relocation - this relocation type is specified by the "P" operand on the appropriate assembler directive or Loader command. It implies that the program segment must begin on a page boundary (i.e. 0,100H,200H,...). Under this option, the Loader places the segment at the next available page boundary after the same segment from the preceding object module or the preceding segment in the case of the Common Segment.

Inpage Relocation - this is specified by the "I" operand on the appropriate assembler directive or Loader command. It implies that the program segment should fit within the current page. If the Loader determines that a program segment cannot fit within the current page, it begins the segment on the next page boundary as though it was "P" relocatable.

In the typical load sequence, the Loader places all Base Segments contiguously in memory; followed by all Program Segments, Data Segments, and the Common Segment. There are no extra bytes between segments. However, if any of the Data Segments specify Page or Inpage relocation, the Loader will start the Data Segment at a page boundary to preserve relocation. To avoid any wasted memory the user can always specify starting addresses. The same problem exists if the Program Segment follows the Base Segment and the Program Segment has specified Page or Inpage

Relocation.

All Common Segments are loaded starting at the same address. The size of the largest Common Segment in the object modules will determine the size of the Common area. Since all Common segments are loaded beginning at the same address, each module specifying the Common Segment should have the same relocation type: byte, page, or inpage. If the relocation types are different, it will be set according to the following precedence: first Page, followed by Inpage, followed by Byte relocatable.

When initially developing and debugging a program, it is sometimes helpful to specify each segment in each assembly as Page relocatable. This then forces the starting address of each module to end in 00H and makes it easier for the user to follow the flow of the program. In this case the assembler output listing contains the correct memory addresses except for an offset that must be added to the high order address byte. Page Relocation may also be specified in the Loader by the PPAGE, DPAGE, or CPAGE commands.

## Base Page

As previously mentioned in both the Assembler and Loader manual, the Base Segment of each module is assumed to reside in the microprocessor memory at addresses from 0 to 0FFH. This enables the assembler to use two byte instructions instead of three byte instructions. The user may actually set the starting address of the Base Segment with a loader command. It is assumed that it would be set to an address within 0-0FFH.

The above assumptions may be invalidated by the user setting a starting Base Segment address greater than 0FFH or the user specifying that the Base Segment is not the first segment with the ORDER command. In this case the Base Segment may reside in addresses larger than 0FFH. Of course, if many modules are being loaded, the length of all Base Segments may be larger than 256 bytes. In any of these cases the Loader will display the message "BASE PAGE OVERFLOW" after the Load Map is printed. This message implies that some part of the Base Segment was forced to reside at addresses greater than 0FFH. The Loading process will continue but the final object module may not run properly. In cases where the Base Segment consists of instructions, or where no references are made to that part of the Base Segment that overflowed the base page, it is possible that the final program will run properly.

# LOADER COMMANDS

The Loader reads a sequence of commands from the Command input device. The commands may be read in an inteactive or batch mode (see Loader Installation Notes). The last command must be either an EXIT or END command.

The object modules are read from the object module input device or from files specified on the LOAD command. The object modules may be read from the same input device as the commands.

The output of the Loader consists of an absolute load module suitable for loading into an actual microprocessor. The output module is written to the object module output device in a format described in the Assembler Manual.

All commands must begin in column 1. Command arguments may begin in any column and must be separated from the commmand by at least one blank. Comments may be placed in the command stream and are indicated by an asterisk in column 1.

The following pages describe the Loader commands. In the command descriptions, brackets, { }, are used to indicate optional arguments. A summary of the commands is given below:

| | |
|---|---|
| PSCT | Set Program Segment Base Address |
| DSCT | Set Data Segment Base Address |
| CSCT | Set Common Segment Base Address |
| BSCT | Set Base Segment Base Address |
| STACK | Set Stack Segment Base Address |
| MEMORY | Set Memory Segment Base Address |
| PPAGE | Set Paging for Program Segment |
| DPAGE | Set Paging for Data Segment |
| CPAGE | Set Paging for Common Segment |
| ORDER | Specify Segment Order |
| START | Specify Starting Output Module Address |
| STKLN | Specify Stack Length |
| NAME | Specify Output Module Name |
| LOAD | Load Specified Object Modules |
| XDEF | Specify Symbol Definitions |
| LIST | List specified Elements |
| NLIST | Do not list specified Elements |
| EXIT | Exit Loader |
| END | End Command stream and finish loading |
| * | Comment |

Command arguments that are numeric may be either decimal or hexadecimal.

Hexadecimal constants are teminated by an "H", e.g. 1FH, and need not have a leading zero even if it starts with the hexadecimal characters A-F.

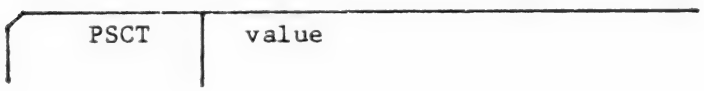
Commands may be read in any order and the same command may be used more than once. The last use of a command determines the command parameters. Commands may be placed before or after the LOAD command except for the PSCT, DSCT, CSCT, BSCT, STACK, and MEMORY commands, which must precede the first LOAD command.

PSCT - Set Program Segment Base Address

The PSCT command is used to specify the starting address of the Program Relocatable Segment. If not specified, the starting address is assumed to be zero, or will begin after the preceding segment if this is not the first segment in memory. This command must be specified before any LOAD commands.

Example:

```
PSCT        400H
```




where:

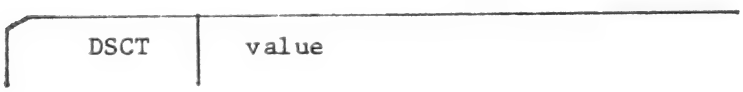
value - specifies the starting address of the Program Segment.

DSCT - Set Data Segment Base Address

The DSCT command is used to specify the starting address of the DATA Relocation Segment. If not specified, the starting address either follows the Program Segment or is assumed to be zero when the Data Segment is the first segment in memory. This command must be specified before any LOAD commands.

Example:

```
DSCT        1000H
```




where:

value - specifies the starting address of the Data Segment.

CSCT - Set Common Segment Base Address

This command is used to specify the starting address of the Common Segment. If not specified, the segment will either start immediately after the preceding segment in memory, or begin at zero if this is the first segment. This command must be specified before any LOAD commands.

Example:

```
CSCT        2048
```

| CSCT | value |
|---|---|

where:

value - specifies the starting address of the Common Segment.

BSCT - Set Base Segment Base Address

This command is used to specify the starting address of the Base Segment. If not specified the segment will either begin at location zero or begin immediately after the preceding segment, when this is not the first segment in memory. This command must be specified before any LOAD commands.

This command is not frequently used, as the Base Segment usually resides within locations 0-0FFH. The user may want to start the Base Segment within 0-0FFH but not at location 0. The user may specify a value greater than 0FFH in this command. This is not considered an error but will cause the "BASE PAGE OVERFLOW" message to be printed during subsequent loading.

Example:

```
BSCT        10H
```

```
BSCT    value
```

where:

value - specifies the starting address of the Base Segment.

STACK - Set Stack Segment Starting Address

This command is used to specify the starting address of the Stack Segment. The length of the Stack Segment is specified by the STKLN command in either the Assembler or Loader. If the Stack address is not specified, it will start immediately following the preceding segment in memory or will begin at zero when this is the first segment. This command must be specified before any LOAD commands.

The 6500 series of microprocessors requires that any stack area reside on page 1 (100H-1FFH). If the Base Address of the Stack Segment does not reside within page 1, the Loader will display the message "STACK NOT ON PAGE 1". This may not be an error for some specific program as the stack may not be used.

Note: the starting address specified by this command is the high address (top) of the Stack Segment.

Example:

```
STACK     1FFH
```

```
STACK     value
```

where:

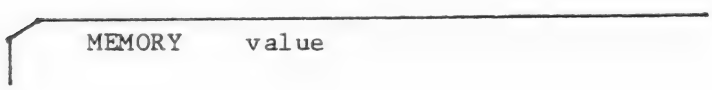
value - specifies the starting address of the Stack Segment.

MEMORY - Set Memory Segment Base Address

The MEMORY command is used to specify the starting address of the Memory Segment. The length of the Memory Segment will be specified as zero on the Load Map but it is actually the length of available memory remaining in a users system after the other segments have been loaded. If not specified, the starting address will be immediately following the preceding segment in memory or begin at zero if this is the first segment. This command must be specified before any LOAD commands.

Example:

```
MEMORY      8000H
```




where:

value - specifies the starting address of the MEMORY Segment.

PPAGE - Set Paging (Relocation Type) for Program Segment

This command may be used to modify the relocation type of a Program segment(s) in the input object modules. As explained under Relocation Types, the assembler indicates to the Loader the relocation type as Byte, Page, or Inpage for each segment in each object module. This command allows the user to override the relocation type specified by the assembler.

The typical use of this command is to allow the user to begin each module on a page boundary, for ease of debugging, and then to specify the final program as byte relocatable to avoid wasted memory space. This command allows the user to change relocation type without reassembling.

This command allows the user to specify that the Program Segment of each module is to be Byte, Page, or Inpage relocatable regardless of the type specified by the Assembler. It also allows the relocation type specified by the Assembler to be used. This command may be changed for each module read by the Loader. The last PPAGE command will be used.

Example:

```
PPAGE      P
```

```
PPAGE     {blank,B,P,I}
```

where:

blank - specifies that the relocation type will be that specified in the Assembler. This is the default.
B - specifies Byte relocation
P - specifies Page relocation
I - specifies Inpage relocation

DPAGE - Set Paging (Relocation Type) for Data Segment

This command may be used to modify the relocation type of a Data Segment(s) in the input object modules. This command is used in the same manner as the PPAGE command and allows the user to specify the Data Segment of each module to be Byte, Page, or Inpage relocatable regardless of the type specified by the assembled.

This command may be changed for each module read by the Loader. The last DPAGE command will be used.

Example:

DPAGE

```
DPAGE     {blank,B,P,I}
```

where:

blank - specifies that the relocation type will be that specified in the Assembler. This is the default.
B - specifies Byte relocation
P - specifies Page relocation
I - specifies Inpage relocation

CPAGE - Set Paging (Relocation Type) for Common Segment

This command may be used to modify the relocation type of Common segment(s) in the input object modules. It allows the user to specify the Common Segment to be Byte, Page, or Inpage relocatable regardless of the type specified by the assembler.

Since all Common Segments are loaded at the same address, the CPAGE command need only be used once at the beginning of the loading process. If the Common Segments of different object modules have different relocation types, the relocation type of the final Common Segment will be set according to the precedence of Page, Inpage, and Byte.

Example:

```
CPAGE     P
```

```
CPAGE     {blank,B,P,I}
```

where:

blank - specifies that the relocation type will be that specified in the Assembler. This is the default.
B - specifies Byte relocation
P - specifies Page relocation
I - specifies Inpage relocation

ORDER - Specify Segment Order

As described under Loader Operation, the normal order of the segments in memory is: Base, Program, Data, Common, Stack, and Memory. The ORDER command is provided for users who do not need to specify starting addresses for each segment but would like the segments to be placed in memory in a different order.

If the user specifies starting addresses for the segments, the order of the segments may be of no particular importance. Keep in mind however, that even if a starting address is specified for a certain segment, any segments following that segment will be loaded at the next subsequent address. Thus if the user wanted say, the Data Segment to reside at address 8000H, and all the other segments to be placed together in lower memory, the ORDER command would have to be specified with the Data Segment being the last argument. If this was not specified, then any segments loaded after the Data Segment would reside in upper memory after the Data Segment.

The user specifies the order of the segments by specifying single letters representing the segments separated by commas. All segments must be specified in the command or an error message is printed and the command is ignored.

Because of the nature of the Base Segment, it will typically be the first segment sepcified in this command.

Example:

```
ORDER      B,D,P,C,S,M
```

```
ORDER      seg,seg,seg,seg,seg,seg
```

where:

seg - specifies one of the six segments types as follows:

- B - Base segment
- C - Common segment
- D - Data segment
- M - Memory segment
- P - Program segment
- S - Stack segment

All six segments must be specified

START - Specify Starting Output Module Address

This command performs no function in the current Loader as the absolute object module format used by MOS Technology does not make a provision for a starting address.
Example:

```
START      7FCH
```

```
START      value
```

where:

value - specifies the starting address be be used in the output object module.

STKLN - Specify Stack Length

The STKLN command is used to specify the length of the Stack Segment to the Loader. If not specified, the stack length is determined by the sum of the stack segment lengths specified in the input object modules.

Example:

```
STKLN      20H
```

```
STKLN      value
```

where:

value - specifies the length of the Stack Segment.

NAME - Specify Output Module Name

The NAME command is used to specify the name of the final output object module. Currently this command performs no function in the output object module, which is in MOS Technology's standard hexadecimal format, and does not contain a name. This directive is included for use when the output object module may be in relocatable format.

The user specified name may be any standard symbol and be up to 6 characters or longer if the symbol length was modified by the user. If the user does not specify a name, the name of the output module will be taken from the first input module.

Example:

```
NAME        READER
```

```
NAME        name
```

where:

name - is a symbol that specifies the object module name.

LOAD - Load specified Object Modules

The LOAD command is used to specify one or more input object modules to be loaded. If the command operand is a number, it is assumed that the input module is to be read from that logical I/O device. If the command operand is not a number, it is assumed the name of a disk file is being specified, and the object module will be read from that file. If any operand is preceded by a minus sign, it indicates that multiple object modules should be read from the specified device or file until an end-of-file (EOF) condition is detected (see Installation Notes). In this case the user need not specify an operand for each object module.

Object modules may be read from a combination of files and I/O devices and may or may not be read until the EOF. The object modules are loaded in the order specified with each module being loaded into memory at a higher address then the preceding module. A user may use as many LOAD commands as needed.

Example:

```
LOAD    7,-FILE1
```

Three modules are to be loaded. The first from unit 7 and the next two from file FILE1 until an EOF.

```
LOAD    module1{,module2,...,moduleN}
```

where:

module - specifies the number of a logical input device or the name of a disk file on which the object module resides. Any module specification preceded by a minus sign will cause object modules to be read until an EOF is detected on the device or file. Operands are separated by commas.

XDEF - Specify Symbol Definitions

This command is used to define and/or change the value of an external definition. If a symbol specified by this command is already an external definition (from an input object module defined by the Assembler XDEF or PUBLIC directives), the value of the symbol is changed to that specified by the user. If the symbol is not already defined, it will be entered into the Loader's symbol table along with the specified value and will then be available to satisfy external references from object modules.

This command allows the user to specify the value of some external symbols at Load time and possibly to avoid a reassembly. To change the value of a symbol that is an external definition in an object module, this command must be specified after the object module has been loaded via the LOAD command. This may be done by specifying this directive immediately before the END command.

Example:

```
XDEF        INPUT=2FH,OUTPUT=200H
```

```
XDEF      sym1=value{,sym2=value, ...,symN=value}
```

where:

sym - is a user defined external definition symbol
value - is the value of the symbol

LIST - List Specified Elements

The LIST command may be used to generate listings of the elements specified. The defaults are: no symbol tables are listed, an object module is produced, no symbols are placed into the output object module, and local symbols are not purged from the input object modules. Local symbols are those placed into the object module by the assembler which are not external definitions. The user should note that placing both external definitions and local symbols into the output object module could cause duplicate symbols to exist in the module. Typically only the local symbols placed into the object modules by the assembler or external definitions will be placed in the output object module. This is the case because the "LIST B" directive in the assembler forms a symbol table which includes both local symbols and external definitions.

Example:

```
                    LIST     T,X        list both local and
                                        definition symbol tables
```

```
LIST      {D,O,P,S,T,X}
```

where:

- D - specifies that external definition symbols will be placed into the output object module.
- O - specifies that an object module is to be produced. (default)
- P - specifies that any symbols present in the input modules (local) be placed into the Loader symbol table. (default)
- S - specifies that the local symbol table be written to the object module and thus may be used for debugging.
- T - specifies that the local symbol table be listed on the list output device.
- X - specifies that the external definition symbol table be listed on the list output device.

NLIST - Do not List Specified Elements

The NLIST command is the opposite of the LIST command and is used to suppress the listing of the elements specified. The elements may be turned back on with the LIST command.

Example:

```
                    NLIST     O         don't produce an
                                        object module
```

```
NLIST     {D,O,P,S,T,X}
```

where:

- D - specifies that external definition symbols will not be placed into the output object module. (default)
- O - specifies that no output module is to be produced. This is useful to check for errors.
- P - specifies that any local symbols present in the input object modules not be placed in the Loader symbol table. This is useful when many modules are being loaded and the symbol table may become full. Of course these local symbols may not be listed in a a symbol table.
- S - specifies that the local symbol table not be written to the object module. (default)
- T - specifies that the local symbol table not be listed on the list output device. (default)
- X - specifies that the external definition symbol table not be listed on the list output device. (default)

EXIT - Exit Loader

The EXIT command is used in the interactive mode to exit the Loader. This command is useful when the user finds an error that will require the leaving the Loader (return to the operating system) to fix. It acts like an END command except that the final load does not take place and an output object module is not produced. This command may also be used in the batch mode by making it the last command in the command stream. In this case the final load will not take place but the object modules and commands will be read and checked for errors.

In the interactive mode, most command errors are recoverable, but errors that occur due to the LOAD command are generally not recoverable.

```
EXIT
```

END - End Command Stream and finish Loader

This command should be the last command in every command stream except if the EXIT command is used as described under that command. It initiates the final steps in the load process.

Comment - Specify Loader Comment

An asterisk may be used to specify a comment in the command stream. The asterisk must be in column one.

Example:

```
* LOADER EXAMPLE
```

# HOW TO USE THE LOADER

## The Loader

The Loader program is usually supplied as an unlabeled, unblocked magnetic tape with 80 character card image records. Other media may be requested.

The Loader is written entirely in FORTRAN and is comprised of a main program and several subroutines. The main program appears first on the tape and the last subroutine is followed by a tape mark. The Loader is located after the assembler on the tape.

The Loader Installation Notes describe program installation and any modifications that may be necessary for a particular computer. It is helpful to read these notes before installing the program.

## Loader Execution

This is a two pass loader in which the commands and object modules are checked for errors during the first pass and a symbol table is formed. Errors detected during this phase of the program will be displayed on the listing. If the user is in the batch mode, any errors found during this pass will cause the loader to terminate with the message "LOAD NOT COMPLETED." If the user is in the interactive mode, only those errors found in the object modules will cause termination of the loading process.

During Pass two of the Loader, the final object module is produced along with a Load Map and a listing of any undefined external references. A symbol table may also be listed.

When executing the Loader, the user should place the Loader commands on the command input device expected by the program. Of particular importance is that the user specify on the LOAD command the correct number of modules to be loaded and where they are located, device or file. It is extremely useful to use the Load until EOF option on the LOAD command if the end-of-file condition can be detected on the particular computer.

## Loader Listing

The following pages show a sample listing from the Loader which is used to describe both the output listing and the loading process. This example is also used as the Loader Test Program.

The first page of the listing shows all commands entered by the user along with any command errors that occur. Following this would be any load module errors that occurred in the modules loaded via the LOAD command. If

no fatal errors occur up to this point, then a Load Map is displayed which lists the names of all input modules followed by the starting addresses of the Base, Program, and Data segments for the final load module . The ending address+1 for each segment is displayed at the end of all modules and is indicated by //. Following this, the starting and ending addresses+1 of the Common, Stack, and Memory segments are displayed. The ending addresses+1 are once again shown by the double slash for these segments. When the starting and ending addresses are the same, the segment has zero length. Following the segment addresses for the relocatable segments, is a list of all absolute segments in the object modules along with their starting and ending addresses. These are listed in numerical order. If certain Loader tables become full it is possible that not all absolute segments will be shown.

Following the Load Map is a list of all external definition symbols as well as local symbols if the user specified the appropriate LIST command option. External Definition symbols are those declared as such in the assembler. Local symbols are those that were output by the assembler if the user had specified the "LIST B" directive in the assembler. These may be used for debugging, but serve no function in the Loader.

After the Load Map, messages may appear which may indicate a user error. The message, "BASE PAGE OVERFLOW" will appear if the ending address of the Base Segment is greater than OFFH. This usually implies that some reference to variables in the Base Segment that were set to 1 byte in the assembler will not reference the correct variable in the final program and thus the output object module will probably not execute properly. The user would typically have to reduce the number of variables in the Base segments in the program. The message "STACK NOT ON PAGE 1" indicates that the final address of the stack area does not reside on Page 1, a requirement of the microprocessor. This is not necessarily an error as the user's program may not even use the stack. Finally the message "SEGMENT OVERLAP" will appear if any of the segments, including any absolute segments, have overlapping addresses.

As shown on the example listing, the only other information that will be displayed on the listing after ths point are any undefined externals found during final load and/or Base segment external references that were declared in the assembler (XREFB directive) whose corresponding value is greater than OFFH and hence is out of range. In this case the value used for the reference will be the low 8 bits, but the user's final program will probably be in error. An undefined external lists the name of the external, the address of the undefined external reference in the original object module and the segment type of the segment from which the reference was made.

The end of the Load Program is indicated by the "LOAD COMPLETED" or "LOAD NOT COMPLETED" message.

**LOADER COMMANDS

```
*
*
* TEST PROGRAM FOR 6500 LINKING LOADER
*
* NOTE, THE OBJECT MODULES ARE READ FROM THE SAME
* DEVICE AS THE COMMAND STREAM.  TO READ THE OBJECT
* MODULES FROM A DIFFERENCT DEVICE, THE LOAD COMMANDS MUST BE
* CHANGED TO THE NEW DEVICE NUMBER.  ALSO IF THE USERS
* COMMAND DEVICE IS NOT 5, THE LOAD COMMAND MUST BE CHANGED.
*
LIST  T,S,X
STACK 1FFH
CSCT  1080H
ORDER B,P,D,S,C,M
STKLN 10H
LOAD  5,5
LOAD  5
END
```

```
**LOAD MAP**

  MODULE     BSCT  PSCT  DSCT

  MAIN       0000  0052  0100
  READ       0051  008C  0100
  MODULE     0052  C0DC  0100
  //         0052  C0ED  010F
  COMMON     1080
  //         1082
  STACK      01EF
  //         01FF
  MEMORY     1082
  //         1082

  ABSOLUTE SEGMENTS

             0400  0403
```

```
**DEFINITION SYMBOLS

CRLF    0081    ECHO    0050    INBUF   0000    INBUFE  C050
READ    008C    TIN     0068    TOUT    0075


**LOCAL SYMBOLS

ASCR    000D    BLNK    0020    BSPA    0006    CCUNT   0051
LENGTH  1080    READ    008C    READ10  0091    READ20  009E
READ30  00AC    READ40  00B4    READ50  00BF    READ60  00C7
READ70  00CC    READ80  00D2    TAB     0009

**MODULE   MAIN
  UNDEFINED EXTERNAL   0010 P - SCAN

**LOAD COMPLETED
```

```
$$
 ASCR    0000DH  BLNK    00020H  BSPA    00008H  CCUNT   00051H
 LENGTH  01080H  READ    0008CH  READ10  00091H  READ20  0009FH
 READ30  000ACH  READ40  000B4H  READ50  000BFH  READ60  000C7H
 READ70  000CCH  READ80  000D2H  TAB     00009H
$$
$1000S2A2FF9A208C00A200B500E8C920D0F9200B5A
$1000E20000E84C5200AD00102902D0F9AD01100567
$100072297F60A8AD00102901D0F98C011060A90688
$1000820D207500A90A20750060A9008551AA200525
$1000926800C918D006208100 4C8C00C90D000A05EA
$1000A29500A551F0E98D801060C97FD00FA5510880
$1000B2F0DDCAC651A9082075004CCC00C909F00890
$1000C204C920B0059500E8E651A551C950F0E20909
$1000D2A550F08B2075004C9100EAAD8010600000788
$0B00E2048D8110100245AAAD010403C2
$0F01C0A900BD0004E8D00320EA00EA0000000529
$0404000506EAEA01E7
$00000C000C
```

LOADER EXAMPLE OUTPUT OBJECT MODULE

## Loader Example

The following pages show three assembly listings of programs that will be combined by the Loader. The actual Load is shown on the preceding pages. The main program contains references to a subroutine called READ and SCAN which are not in the program but which are declared external in another object module. The second assembly listing shows the READ routine which is required by the Main program and also shows that the READ routine requires the I/O drivers TIN and TOUT which are declared external in the Main Program. The third program cotains no links to the other programs but does contain some absolute code. The Common segment is also used to provide linkage between the programs.

The command stream on the preceding pages shows that the user has specified the starting addresses of the Common and Stack Segments. The ORDER command is used to place the segments in an order different than the standard order. The LIST command is used to obtain a symbol table listing of both local and external definition symbols as well as place the local symbols into the output object module. Finally the LOAD command is set to read the three modules from the device shown.

The Load Map shows the starting and ending addresses of the three modules in the order loaded. Note that the third module has a "DSCT P" directive in the assembly and consequently the load map shows that the Data Segment for this module starts on the next page boundary.

An undefined external is listed for the Main module. The external's address, relocation type, and name is specified. It can be seen that SCAN is not in any module. The user could have specified the address of the routine with a XDEF command.

Finally the symbol table of all definition and local symbols used in the program along with their absolute values is listed. The user can determine from the addresses as well as the final object module, displayed on a subsequent page, that the modules have been linked together correctly to form a final absolute module with all addresses adjusted to the correct values and all links between modules resolved.

Following the above example, a Loader run is displayed that contains a few errors. Most of the load errors shown will not occur except under unusualy conditions, and they have been shown for illustrative purposes only.

The final absolute object module from the example is shown along with the local symbols that were placed in the module.

```
                                        NAME    MAIN
                                        LIST    X
                                        XDEF    INBUF,INBUFEND,TIN,TOUT,CRLF,ECHO
                                        XREF    READ,SCAN
                                *
                                *
                                * THIS IS A SAMPLE PROGRAM THAT SHOWS MOST OF THE FEATURES OF THE
                                * RELOCATABLE ASSEMBLER AND LOADER.  THREE MODULES ARE LINKED
                                * TOGETHER TO FORM THE FINAL PROGRAM.  EXTERNAL DEFINITIONS AND
                                * REFERENCES AS WELL AS COMMON ARE USED TO PERFORM THE LINK.
                                *
                                * BELOW IS THE MAIN PROGRAM AND THE I/O DRIVERS.  THIS IS
                                * LINKED TO A ROUTINE WHICH READS A LINE OF CODE AND WHICH
                                * ITSELF REQUIRES THE I/O DRIVERS
                                *
                                *
                                        BSCT
0000                    INBUF   RES     80              INPUT BUFFER
0050                B   INBUFEND =      *               END OF BUFFER
0050                    ECHO    RES     1               ECHO FLAG
                                *
                                        PSCT                    ;SET PROGRAM SEGMENT
0000 A2 00          S   MAIN    LDX     #.LOW.STACK
0002 9A                         TXS                     SET STACK POINTER
0003 20 00 00       E           JSR     READ            READ NEXT LINE
0006 A2 00                      LDX     #0              INDEX INTO BUFFER
0008 B5 00          R   MAIN10  LDA     INBUF,X         GET NEXT CHARACTER
000A E8                         INX
000B C9 20                      CMP     #BLNK           CHECK FOR NON BLANK
000D D0 F9                      BNE     MAIN10
000F 20 00 00       E           JSR     SCAN            GET VALUE
0012 E8                         INX
0013 4C 00 00       R           JMP     MAIN
                                *
                                * NAME - INB
                                *
                                * THIS ROUTINE WILL INPUT A CHARACTER FROM THE TERMINAL
                                *
                                * ENTRY PARAMETERS
                                *   NONE
                                *
                                * EXIT PARAMETERS
                                *   A         - INPUT CHARACTER
                                *
                                *
0016 AD 00 10               INB     LDA     USTAT           READ UART STATUS
0019 29 02                          AND     #RRDY           CHECK IF READY
001B D0 F9                          BNE     INB             NOT READY YET
001D AD 01 10                       LDA     UDATIN          READ DATA
0020 29 7F                          AND     #127            DELETE PARITY BIT
0022 60                             RTS
                                *
                                * NAME - OUTB
                                *
```

```
                                * THIS ROUTINE IS USED TO OUTPUT A CHARACTER TO
                                * THE TERMINAL
                                *
                                * ENTRY PARAMETERS
                                *   A       - CHARACTER TO OUTPUT
                                *
                                * EXIT PARAMETERS
                                *   NONE
                                *
                                *
0023 A8                         OUTA    TAY
0024 AD 00 10                   OUTB1   LDA     USTAT
0027 29 01                              AND     #TRDY           CHECK IF READY
0029 D0 F9                              BNE     OUTB1           NOT READY
002B 8C 01 10                           STY     UDATOUT         OUTPUT DATA
002E 60                                 RTS
                                *
                                * NAME - CRLF
                                *
                                * THIS ROUTINE OUTPUTS A CARRIAGE RETURN AND LINE FEED
                                *
                                *
002F A9 0D                      CRLF    LDA     #ASCR
0031 20 23 00               P           JSR     OUTB
0034 A9 0A                              LDA     #ASLF
0036 20 23 00               P           JSR     OUTB
0039 60                                 RTS
                                *
                                *
1000                            USTAT   EQU     1000H           UART STATUS
1001                            UDATOUT EQU     1001H           UART OUTPUT
1001                            UDATIN  EQU     1001H           UART INPUT
0001                            TRDY    EQU     1               TRANSMIT READY
0002                            RRDY    EQU     2               RECEIVE READY
000D                            ASCR    EQU     13
000A                            ASLF    EQU     10
0020                            BLNK    EQU     20H
0016                        P   TIN     EQU     IN8
0023                        P   TOUT    EQU     OUTB
003A                            END     MAIN
```

ASSEMBLER ERRORS = 0

CROSS REFERENCE

| LABEL | | VALUE | REFERENCE | | | |
|---|---|---|---|---|---|---|
| ASCR | | 000D | 77 | -89 | | |
| AS_F | | 000A | 79 | -90 | | |
| BLNK | | 0020 | 29 | -91 | | |
| CRLF | P | 002F | 3 | -77 | | |
| ECHO | B | 0050 | 3 | -20 | | |
| INB | P | 0016 | -46 | 48 | 92 | |
| INBUF | B | 0000 | 3 | -18 | 27 | |
| INBUFE | B | 0050 | 3 | -19 | | |
| MAIN | P | 0000 | -23 | 33 | 94 | |
| MAIN10 | P | 0008 | -27 | 30 | | |
| MEMORY | M | 0000 | 0 | | | |
| NARG | | 0000 | 0 | | | |
| OUTB | P | 0023 | -65 | 78 | 80 | 93 |
| OUTB1 | P | 0024 | -66 | 68 | | |
| READ | E | 0000 | 4 | 25 | | |
| RRDY | | 0002 | 47 | -88 | | |
| SCAN | E | 0001 | 4 | 31 | | |
| STACK | S | 0000 | 0 | | | |
| TIN | P | 0016 | 3 | -92 | | |
| TOUT | P | 0023 | 3 | -93 | | |
| TRDY | | 0001 | 67 | -87 | | |
| UDATIN | | 1001 | 49 | -86 | | |
| UDATOU | | 1001 | 69 | -85 | | |
| USTAT | | 1000 | 46 | 66 | -84 | |

```
023E0006MAIN**0000013A0003020000030300006304910003050006030600006BB
18160C06READ**0006SCAN**0035
162E0C01160006TIN***002F0006CRLF**00230006TOUT**0C35
162E0C04500006ECHO**C0000006INBUF*00500006INBUFE00E4
06180001000A2009A200000A200E3
240A0005010100CB
200C00030000400CD
064000010800B500E8C920D0F9200000E84C0000AD0010290200F9AD0110297F60ABEF
220800031400BF
240A0004030900C2
200C00030100100CC0
06340001240AD00102901D0F98C011060A90D202300A90A2023006CA5
220C00033200370066
040A0001010000F0
0E020CF0
```

MAIN OBJECT MODULE

```
                                        NAME    READ
                                        LIST    X
                                        LIST    B               GET SYMBOL TABLE
                                        XDEF    READ
                                        XREF    CRLF,TIN,TOUT
                                        XREFB   INBUF,IBUFEND,ECHO
                                *
                                        BSCT
0000                            COUNT   RES     1               BUFFER COUNT
                                *
                                        PSCT                    SET PROGRAM SEGMENT
                                *
                                * NAME - READ
                                *
                                * THIS ROUTINE READS IN A LINE FROM THE TERMINAL AND
                                * PLACES IT INOT THE INPUT BUFFER.  THE FOLLOWING ARE
                                * SPECIAL CHARACTERS
                                *   CR        - END OF LINE
                                *   CONTROL X - DELETE CURRENT LINE
                                *   DEL       - DELETE LAST CHARACTER
                                * ALL DISPLAYABLE CHARACTERS BETWEEN BLANK AND Z AND THE
                                * ABOVE SPECIAL CHARACTERS ARE RECOGNIZED BY THIS
                                * ROUTINE AS WELL AS THE TAB.  ALL OTHERS CHARACTERS ARE
                                * IGNORED.  AN ATTEMPT TO INPUT MORE CHARACTERS THEN IS
                                * ALLOWED IN THE INPUT BUFFER WILL BE INDICATED BY A BACKSPACE.
                                *
                                * ENTRY PARAMETERS
                                *   ECHO      - ECHO FLAG, 0 = NO ECHO
                                *
                                * EXIT PARAMETERS
                                *   INBUF     - CONTAINS INPUT LINE
                                *
                                *
0000 A9 00                      READ    LDA     #0
0002 85 00              B               STA     COUNT           SET CHARACTER COUNT
0004 AA                                 TAX                     INPUT BUFFER INDEX
0005 20 00 00           E       READ10  JSR     TIN             READ NEXT CHARACTER
0008 C9 18                              CMP     #24             CHECK FOR CONTROL X
000A D0 06                              BNE     READ20
000C 20 00 00           E               JSR     CRLF
000F 4C 00 00           E               JMP     READ            START AGAIN
0012 C9 0D                      READ20  CMP     #ASCR
0014 D0 0A                              BNE     READ30          NOT A CARRIAGE RETURN
0016 95 00              E               STA     INBUF,X         PUT CR AT END OF LINE
0018 A5 00              B               LDA     COUNT
001A F0 E9                              BEQ     READ10          NO INPUT YET
001C 8D 00 00           C               STA     LENGTH          SAVE LINE LENGTH
001F 60                                 RTS
0020 C9 7F                      READ30  CMP     #127            CHECK FOR DELETE
0022 D0 0F                              BNE     READ50
0024 A5 00              B               LDA     COUNT           GET COUNT
0026 F0 D3                              BEQ     READ10          NO CHARACTERS YET
0028 CA                         READ40  DEX                     DECREMENT POINTER
0029 C6 00              B               DEC     COUNT           DECREMENT COUNT
```

```
      55 002B A9 08                  LDA     #BSPA
      56 002D 20 00 00     E         JSR     TOUT        BACKSPACE
      57 0030 4C 40 00     P         JMP     READ70
      58 0033 C9 09          READ50  CMP     #TAB        CHECK FOR A TAB
      59 0035 F0 04                  BEQ     READ60
      60 0037 C9 20                  CMP     #BLNK
      61 0039 B0 05                  BCS     READ70
      62 003B 95 00        E READ60  STA     INBUF,X     PUT CHARACTER INTO BUFFER
      63 003D E8                     INX
      64 003E E6 00        B         INC     COUNT       INCREMENT COUNT
      65 0040 A5 00        B READ70  LDA     COUNT       GET COUNT
      66 0042 C9 50                  CMP     #80         CHECK IF FULL
      67 0044 F0 E2                  BEQ     READ40      YES
      68 0046 A5 00        E READ80  LDA     ECHO        GET ECHO FLAG
      69 0048 F0 B8                  BEQ     READ10      DONT ECHO
      70 004A 20 00 00     E         JSR     TOUT
      71 004D 4C 05 00     P         JMP     READ10
      72                             *
      73                             CSCT
      74 0000                LENGTH  **+1                LINE LENGTH
      75                             *
      76                             *
      77                             *
      78 000D                ASCR    EQU     13
      79 0008                BSPA    EQU     8
      80 0020                BLNK    EQU     20H
      81 0009                TAB     EQU     9
      82 0001                        END
```

ASSEMBLER ERRORS = 0

CROSS REFERENCE

| LABEL | | VALUE | REFERENCE | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| ASCR | | 0000 | 42 | -78 | | | | | |
| BLNK | | 0020 | 60 | -80 | | | | | |
| BSPA | | 0008 | 55 | -79 | | | | | |
| COUNT | B | 0000 | -9 | 35 | 45 | 51 | 54 | 64 | 65 |
| CRLF | E | 0000 | 5 | 40 | | | | | |
| ECHO | E | 0005 | 6 | 68 | | | | | |
| IBUFEN | E | 0004 | 6 | | | | | | |
| INBUF | E | 0003 | 6 | 44 | 62 | | | | |
| LENGTH | C | 0000 | 47 | -74 | | | | | |
| MEMORY | M | 0000 | 0 | | | | | | |
| NARG | | 0000 | 0 | | | | | | |
| READ | P | 0000 | 4 | -34 | 41 | | | | |
| READ10 | P | 0005 | -37 | 46 | 52 | 69 | 71 | | |
| READ20 | P | 0012 | 39 | -42 | | | | | |
| READ30 | P | 0020 | 43 | -49 | | | | | |
| READ40 | P | 0028 | -53 | 67 | | | | | |
| READ50 | P | 0033 | 50 | -58 | | | | | |
| READ60 | P | 003B | 59 | -62 | | | | | |
| READ70 | P | 0040 | 57 | 61 | -65 | | | | |
| READ80 | P | 0046 | -68 | | | | | | |
| STACK | S | 0000 | 0 | | | | | | |
| TAB | | 0009 | 58 | -81 | | | | | |
| TIN | E | 0001 | 5 | 37 | | | | | |
| TOUT | E | 0002 | 5 | 56 | 70 | | | | |

```
023E0006READ**0006[illegible]
183E0006CRLF**0006TIM***0006TOUT**0006INBUF*0106IBUFEN0106ECHD**0155
16120001C00006READ**008D
123C0000800068SPA**002000068LNK**000D0006ASCR**0CC90006TAB***0039
123C0001000006READ**00050006READ100012000 6READ200C200006READ300046
123C0001280006READ4000330006READ50003B0006READ6000400006READ700095
12120001460006READ800041
12120003000006LENGTH0055
12120004000006COUNT*0057
06400001 0000A9006500AA200000C918D006200000 4C0000C90DD00A9500A500F0E9DB
220800031000C3
240E0004 03003001900AB
201C0003010006000000 0D000340170053
061800011C008D000060C97FD00FB1
240A0003031D00AF
06400001 2400A500F0DDCAC600A9062000004C4000C909F004C920B0059500E8E6006F
220800033100A2
24120004 0325002A003F0035
20140003 02002E0003403C0D1A
06280001 4000A500C950F0E2A500F0BB2000004C050C40
220800034E0085
240A0004 0341008A
20140003 05404700020048 00F0
040A0000 10000F1
0E020CF0
```

READ OBJECT MODULE

```
ERR LINE  ADDR  B1 B2 B3 B4                    6500 ASSEMBLER VER 1.0MR                    PAGE    1

    1                                          LIST      X
    2                                          PSCT
    3  0000  EA                               NOP
    4  0001  AD 00 00       C                 LDA       LEN1
    5  0004  6D 00 04                         ADC       DATA
    6  0007  8D 01 00       C                 STA       LEN2
    7  000A  10 02                            BPL       LAB1
    8  000C  45 AA                            EOR       0AAH
    9  000E  AD 01 04                LAB1     LDA       DATA+1
   10                                *
   11                                          DSCT      P                 SET DATA SEGMENT
   12  0000  A9 00                            LDA       #0
   13  0002  BD 00 04                         LDA       DATA,X
   14  0005  E8                               INX
   15  0006  D0 03                            BNE       SKP
   16  0008  20 0E 00       P                 JSR       LAB1
   17  000B  0E 00           P   SKP         .WORD     LAB1
   18  000D  00 00                           .WORD     .LOW.DATA
   19                                *
   20                                          ASCT
   21                                          ORG       400H
   22  0400  05 06                   DATA     BYTE      5,6,.LOW.LAB1
   23  0402  0E                P
   24  0403  EA                               NOP
   25                                *
   26                                          CSCT
   27  0000                          LEN1     RES       1
   28  0001                          LEN2     RES       1
   29  0002                                   END


    ASSEMBLER ERRORS =    0
```

CROSS REFERENCE

| LABEL | | VALUE | REFERENCE | | | | |
|---|---|---|---|---|---|---|---|
| DATA | | 0400 | 5 | 9 | 13 | 18 | -22 |
| LAB1 | P | 000E | 7 | -9 | 16 | 17 | 22 |
| LEN1 | C | 0000 | 4 | -27 | | | |
| LEN2 | C | 0001 | 6 | -28 | | | |
| MEMORY | M | 0000 | 0 | | | | |
| VARG | | 0000 | 0 | | | | |
| SKP | D | 0008 | 15 | -17 | | | |
| STACK | S | 0000 | 0 | | | | |

```
023E0006MODULE000001110003020F000203020003040000030500000306000003F0
062A00010000EAAD000060000480010010024 5AAAD010486
240E0003030200080 08E
062600020000A9008D0004E8D003200E000E00000071
240E000103090008 0086
0610000000040 5060EEAE3
240A0001010204CA
040A00000 10000F1
0E0200F0
```

MODULE OBJECT MODULE

```
                6500 LINKING LOADER VER 1.0

  **LOADER COMMANDS


*
*
* TEST PROGRAM FOR 6500 LINKING LOADER
*
* NOTE, THE OBJECT MODULES ARE READ FROM THE SAME
* DEVICE AS THE COMMAND STREAM.  TO READ THE OBJECT
* MODULES FROM A DIFFERENCT DEVICE, THE LOAD COMMANDS MUST BE
* CHANGED TO THE NEW DEVICE NUMBER.  ALSO IF THE USERS
* COMMAND DEVICE IS NOT 5, THE LOAD COMMAND MUST BE CHANGED.
*
LIST  T,S,X
STACK 1FFH
CSCT  1080H
ORDER B,P,D,S,C,M
STKLN 10H
LOAD  5,5


**MODULE   MAIN
   RECORD OUT OF SEQUENCE
   RECORD    5 - 240A00050101000C8
LOAD  5


**MODULE
   HEADER RECORD ERROR
   RECORD    1 - 062A00010000EAAD00006D00048D0100100245AAAD010486
   RECORD OUT OF SEQUENCE
   RECORD    3 - 240E000303020009008E
END


  **LOAD NOT COMPLETED
```

## APPENDIX A

## LOADER MESSAGES

Messages from the Loader may be classified into Command Error Messages and Load Messages. Command errors are due to invalid commands or command parameters and always cause termination of the loading process in the batch mode. Command messages are listed beneath the actual command. Load messages occur during the loading of object modules initiated by the LOAD command. These messages may be fatal or informative. For most load messages, the message is followed by the record number in the input module and the actual record in error. For a particular module the module name is also listed at the start of the messages.

Most load errors should not occur. If they do, the user is advised to first reassemble the program then to reload. If the error persists, the user should contact Microtec.

### Command Messages

Invalid Command - a command specified by the user is not a legal Loader command.

Invalid Operand - an operand specified for a command contains invalid characters, does not exist, or is too large.

Command Not Allowed - this command is not allowed at this point in the program. Due to specifying a load address after a LOAD command has been specified.

Symbol Table Full - user specified a XDEF command and no more room exists in the symbol table.

Module Greater than 64K - at final load time the lengths of all programs segments is greater than 64K, the maximum memory size.

File Not Found - a file specified on the LOAD command does not exist or possibly an invalid LOAD command operand was specified.

Invalid Symbol - a XDEF command is specified that contains an invalid symbol.

## Load Messages

Invalid Hex Character - a character in the record shown contains an invalid hexadecimal character. Some records contain symbols as well as hexadecimal numbers. This message does not apply to those symbols.

Invalid Checksum - the record has a checksum error and probably contains some changed characters. Possibly a computer hardware error.

Header Record Error - a header record was not the first record in the object modue or a header record was found after the first record.

Record too large - a record specifies a record length that is greater than 72 characters.

Invalid Record Type - a record specifies a record type that does not exist in the Loader.

Invalid ID or Type - some internal parameters on this record are illegal.

Address out of Range - a relocation record specifies relocation at an address outside the range of relocation specified on the header record or on the preceding record. Possibly records are out of sequence.

External Index out of Range - an External Reference is made to an external symbol that does not exist in this object module.

External Table Full - current object module specifies more external symbols then may be contained in external table. Increase size of table.

Record Out of Sequence - an object module record was read that is out of sequence in the module or the user may have inadvertenly mixed the records if they are on cards.

Symbol Table Full - an external definition object module record is being processed and the symbol table is full.

Undefine External - a reference is made to an external symbol that has not been defined in another module or by the user. The address of the external reference in the original module is listed along with the symbol name and segment.

Duplicate Definition - an external definition is defined that was already defined in another module. Loading will continue and the symbol will be listed.

Module Greater than 64K - during initial loading, the sum of all segment lengths exceeds the 64K memory size.

Segment Overlap - due to user specified addresses, or absolute segments, one or more of the segments overlap. This is an informative message and loading continues.

External Not Base Page - a symbol declared as Base Page external (XREFB) in the assembler has a value that is greater than OFFH in the Loader. The name of the external is listed. Loading continues with the low 8 bits of the value being used.

Base Page Overflow - the end address of the Base segment exceeds OFFH. Any references to variables beyond the point of overflow will cause the final program to be wrong and to run incorrectly. Reassemble the programs to reduce Base Page usage.

Stack not on Page 1 - the final address of the Stack segment does not reside on Page 1. This is an informative messsage as the user may not even use the stack.

# LOADER INSTALLATION NOTES

These notes are designed to help the user install the Loader and make any modifications needed for a particular computer. The notes are separated into six sections: Program Installation; Program Modifications; Batch/Interactive Mode; Program I/O; Memory Requirements and Overlays; and NOVA Modifications.

## A. Program Installation

1. Do not hesitate to contact Microtec to discuss any Installation problems or changes to the program. Because we have experience in the installation of our programs on most computers, we can advise the user of any potential problems.

2. The Loader should be compiled once and its object module stored on some secondary device (disk). Compile the program in the usual manner, assigning it a name which can be referenced by an Execute or Run command for the computer. If upon loading the compiled program it is discovered that insufficient main memory is available to hold the entire program, refer to the section describing Overlay Structures.

3. The user may compile the program as one entire program or as individual subroutines. It is easier to make changes to the program when each subroutine is compiled separately and linked together.

4. It is recommended that the user install and test the program as received from Microtec before making any changes to the program. It is desirable to defer incorporating program updates (if any) until the delivered program is installed and tested.

## B. Program Modifications

1. The variable IBIT corresponds to the number of bits per word in the host computer. IBIT is initially set to 16. This variable determines how may characters are packed into one host computer word for symbols stored in the Loader symbol table. The user may want to increase this variable if the host computer has a longer word length. Increasing IBIT will allow a larger number of symbols to be stored in a fixed amount of memory. When initially installing the program, it is suggested that IBIT be left at 16 until the program is known to be operating correctly.

2. To increase the size of the symbol table and consequently the number and length of the symbols the symbol table can hold, the user must change certain variables. The variables that must be changed depend on: the number of bits per host computer word (see 1.), the number of symbols in the symbol table, and the number of characters used to define a symbol. The variables that define these parameters are as follows:

```
IBIT  - number of bits per host computer word (set by user)
MLAB  - maximum label length in characters (set by user)
ICCNT - number of characters per host computer word (calculated)
IWORD - number of computer words per symbol (calculated)
LTAB  - length of symbol table (set by user)
```

The user must change the following variables to reflect the size of the symbol table and length of a symbol. The length of a symbol should correspond to the length set in the Assembler. The arrays are in COMMON, and therefore, the dimensions need to be changed in every Subroutine.

```
ITAB(IWORD,LTAB)        where: IWORD = (MLAB-1)/ICCNT
ITABV(LTAB)                    ICCNT = IBIT/8
ITABS(LTAB)
NAME(WORD)
```

C. Batch/Interactive Mode

1. The program is delivered with the Batch/Interactive flag, IBAT, set for batch operation. In the Batch mode, commands are echoed to the listing device and all command errors are fatal, the final load does not occur. In the Interactive mode, commands are not echoed to the listing device, and some errors become non-fatal. The only fatal command errors are those that may cause some object modules to be loaded before an error is found on the LOAD command line. This variable may be set in Subroutine INIT to the Interactive mode.

D. Program Input/Output

1. The logical I/O device assignments assumed in the Loader are:

```
IPCH  = 4 (object module output device)
ICRD  = 5 (command input device)
IPRT  = 6 (output listing device)
IMFLE = 7 (intermediate file, disk or tape)
IFIL  = 18 (input object module disk file number; when
           an input object module is on a file, the file
           name is equated to IFIL)
IRDR  = __ (set dynamically during program execution to
           the input device specified by the LOAD command)
```

These device assignments may have to be changed for your system. This may be done in either the Job Control Stream or in the program itself. If the device assignments are to be changed in the program, the variables may be found in Subroutine INIT.

2. Reading from and writing to a bulk storage device such as a disk is not standard in FORTRAN. See the Assembler Installation Notes for a discussion of the various methods.

3. All Program I/O activity except for generation of the output

listing is done in Subroutine INOUT. This includes the reads and writes of the Intermediate files, reading the command input, reading the object module input, and writing the output object module.

4. There are alternative ways of passing relocatable object modules from the Assembler to the Loader (see discussion in Assembler Notes). The Input devices or files that hold the object modules to be loaded by the Loader are specified as LOAD command arguments. When a disk file is specified as an argument, Subroutine EQUAT is used to associate the disk file name to the logical device, IFIL (see 1. above), so that the file may be read by the input statements in subroutine INOUT. There are two basic sections to the EQUAT subroutine. First, the file name is packed into a contiguous Hollerith string. The code used to pack the characters of the file name into a string will work on any two's complement computer. For a one's complement computer, one line of code must be changed. The required change is marked with comments in Subroutine EQUAT. Two variables in Subroutine INIT must be set to the correct values for EQUAT to work properly. These are as follows:

ISBIT - actual number of bits in computer word. This may or may not be the same as IBIT.
ICHBT - number of bits per host computer character.

The place to change these in INIT are marked with comments.

The second part of Subroutine EQUAT consists of the code required to open the named disk file and associate it with the logical device number, IFIL. This code usually consists of one statement. The CALL ASSIGN statement that currently exists in the program is for a PDP-11. As mentioned in the Assembler Notes, some computers can read disk files without any special code to open the file. In this case Subroutine EQUAT may not be needed. The user will have to check their computer manuals to find out what statements are required to perform the above function. A simplified EQUAT routine for a PDP-11 is listed at the end of the Installation notes.

5. Refer to the section on Input/Output in the Assembler Installation Notes, as many of the points discussed there apply to the Loader.

6. The I/O statements needed to read in an object module disk files may be different depending upon whether the module is read from an I/O device or a file. The statements in Subroutine INOUT at line number 200 have two I/O statements, one for reading from a file and one from a device. For most machines these statements will be the same as shown. Some users may have to change one or the other. Comments in INOUT describe the necessary changes.

7. Statements are included in Subroutine INOUT for handling an end-of-file (EOF) condition on both the command and object module inputs. A typical READ statement with the END condition specified as shown in INOUT can be used on most computers. Some computers require a call to a system routine or some other statement to detect the EOF. Comments in Subroutine INOUT are included to show where the program

should branch if an EOF is detected. The user may not use the LOAD command with the minus sign option if an EOF cannot be detected. This is because the minus sign option causes object modules to be read from a file or device until the EOF is encountered. Note: no modifications need be made to the INOUT routine if EOF's are not detected.

E. Memory Requirements, Overlays, and Chaining

1. The Loader program is smaller than the Assembler program. Overlaying should not be necessary. However, for users who may want to form their own overlays or to Segment their programs, the following list shows each routine in the Loader and all the routines that call it.

```
MAIN   -
INIT   - MAIN
INOUT  - MAIN,COMM,OBJ,OUT
COMM   - MAIN
OBJ    - MAIN
LABEL  - COMM,OBJ
SYMBL  - COMM,OBJ,LABEL
SCAN   - COMM
NAMES  - COMM,OBJ,OUT
COMIN  - MAIN
OUT    - OBJ
HEXIN  - OBJ
AHEX   - COMM,NAMES,ERROR
EQUAT  - MAIN
ERROR  - MAIN,COMM,OBJ
```

The COMM and OBJ subroutines are the largest subroutines in the program. They are approximately the same size and thus can be easily overlayed. Both are called only from the Main Program.